CÓDIGOS DE RED APLICADOS SOBRE MÚLTIPLES FLUJOS DE DATOS EN TRANSMISIÓN MULTICAST

TESIS DOCTORAL

Monografía presentada para obtener el título de
Doctor por la Universidad del Norte de Barranquilla
en el Programa de Doctorado en Ingeniería de Sistemas y Computación

José Duván Márquez Díaz

Tutor
Ismael Gutiérrez García

Co-tutor
Elías Niño Ruíz

Programa de Doctorado en Ingeniería de Sistemas y Computación
Departamento de Ingeniería de Sistemas y Computación
División de Ingenierías
Universidad del Norte

Barranquilla, Junio de 2018

CÓDIGOS DE RED APLICADOS SOBRE MÚLTIPLES FLUJOS DE DATOS EN TRANSMISIÓN MULTICAST

TESIS DOCTORAL

Monografía presentada para obtener el título de
Doctor por la Universidad del Norte de Barranquilla
en el Programa de Doctorado en Ingeniería de Sistemas y Computación

José Duván Márquez Díaz

Ismael Gutiérrez García
Vo.Bo. Tutor

Co-tutor
Elías Niño Ruíz

Programa de Doctorado en Ingeniería de Sistemas y Computación
Departamento de Ingeniería de Sistemas y Computación
División de Ingenierías
Universidad del Norte

Barranquilla, Junio de 2018

A mis hijos, Miguel y Camilo.

Agradecimientos

"Y así fue: Vio entonces Dios todo lo que había hecho, y todo era muy bueno"
Génesis1:31

Este trabajo de grado, no es una meta, sino el camino hacia más elementos que me hagan cultivar la esperanza de un mañana mejor a través de mi profesión. Un mañana que nos haga soñar con un mundo bueno y mejor como nuestro Creador lo desea.
Gracias al Espíritu Santo por haberme asistido durante el desarrollo de este trabajo. Gracias en cada idea y cada situación que me ayudó a resolver.
Gracias a mis hijos, Miguel y Camilo, por ser el combustible para trabajar en este proyecto.
Un lugar especial ocupan mis padres: Elba y José, que desde sus capacidades, contribuyeron con su motivación constante. Igualmente, mis hermanos, Iván y Mónica, que me dieron su gran apoyo y fortaleza en los momentos difíciles.
Un agradecimiento muy especial a mi tutor, Ismael Gutiérrez, sin su apoyo en la conceptualización matemática, hubiera sido más difícil entender las bases de este trabajo.
A la Oficina de Desarrollo Profesoral y Uninorte, agradezco su apoyo por becarme una vez más para hacer lo que más me gusta: estudiar.
A DTIC, encabezado por mi gran amigo de batallas académicas universitarias: Danilo, y su equipo de trabajo: Henri, Noelia y Andrés. Ellos fueron mi soporte técnico con la VPN y Matlab.
Igualmente, agradezco la gran ayuda con ideas, discusiones, consensos y disensos, establecidos con el grupo de estudiantes de pregrado a quienes puse a pensar en Network Coding: Eric, José Daniel, Néstor, Yezid, Sebastián, Melanis, Sahara, José Rafael, y Carolina.
Un merecido agradecimiento a Ana Lu y Javier, quienes desde la Universidad Complutense, siempre me apoyaron para concluir exitosamente mi doctorado y en especial mi trabajo de grado.
Un agradecimiento especial, a Marle, Carlos, Alfonso y Rocío, del cuerpo de profesores de Sistemas, quienes han sido un gran soporte en la realización de este trabajo con su motivación y conocimientos.
Un gran agradecimiento a alguien, que ajena a todas las lides de Sistemas y Computación, fue mi gran apoyo moral y espiritual, en los momentos difíciles que viví durante la mitad de este trabajo, y que me ha acompañado con su motivación y soporte: Mónica.
Gracias a todos los que me tuvieron paciencia, a los que dejé de ver, de saludar y de llamar. A aquellos con los que no podía compartir mucho tiempo por estar trabajando en este proyecto.

Resumen

Las redes de enrutamiento multicast se establecen para la transmisión desde un nodo fuente emisor hacia un conjunto de nodos sumideros receptores. Sin embargo, estas redes en su proceso de enrutamiento pueden repetir paquetes por algunos enlaces y, además, pueden retrasar la transmisión, ya que deben crear colas de espera en las interfaces de salida hacia los nodos destinos. Los nodos que las conforman, trabajan con la política de almacenamiento y reenvío, de tal modo que los paquetes deben reenviarse, en el mejor de los casos, en el orden en que ingresan al nodo enrutador. Durante el proceso de configuración de la red de enrutamiento multicast, se pueden conformar varios enlaces de salida desde el nodo fuente, conllevando la llegada de múltiples paquetes simultáneamente a los nodos de enrutamiento intermedio, generando estos retrasos. La presencia de Network Coding en conjunto con el enrutamiento multicast, permite que dentro de estos nodos intermedios, de igual manera que en el nodo fuente y los sumideros, se pueda llevar a cabo la combinación lineal de los paquetes que ingresan, produciendo un paquete de salida codificado de igual tamaño que los entrantes, aumentando el uso del ancho de banda por la no espera en los buffers de las interfaces de salida. Con Network Coding deben ser solucionados dos problemas con enrutamiento multicast. El primero es hallar la red multicast de mínimo flujo máximo común que alcance al conjunto de sumideros. El segundo es la búsqueda de un esquema de solución a un sistema lineal de ecuaciones, para la entrega de los paquetes, enviados simultáneamente desde el nodo fuente. Sin embargo, hallar la red multicast a partir de una red general de comunicaciones, y que pueda responder a la codificación de paquetes, no es tarea fácil, dada la naturaleza arbitraria de la red. En primer lugar, en este trabajo de tesis, se proponen tres algoritmos que configuran una red multicast de comunicación, tendiente a tener solución para la recepción del mínimo flujo máximo de paquetes enviados simultáneamente. Los algoritmos están basados en el método de Ford-Fullkerson con BFS y DFS, y en un algoritmo de búsqueda exhaustiva de caminos disyuntos. En segundo lugar, se propone un método para encontrar, desde el nodo fuente del grafo multicast, el orden de salida de los paquetes que conforman el mínimo flujo máximo, y que permita su decodificación en los nodos sumideros.

Palabras clave: Algoritmos, ancho de banda, búsqueda en anchura, búsqueda en profundidad, camino, camino aumentado, caminos disyuntos, campo finito, codificación de paquetes, codificación lineal de red, colisión, combinación lineal de paquetes, corte, cuello de botella, decodificación de paquetes, diferencia simétrica de conjuntos, enlaces entrantes, enlaces salientes, enrutador, espacio vectorial, flujo máximo, grafo de comunicaciones, grafo dirigido acíclico, grafo lineal dirigido etiquetado, grafo multicast de mínimo flujo máximo, matriz de adyacencia, mínimo flujo máximo, nilpotencia, nodo codificador, nodo fuente, nodo sumidero, orden de paquetes, paquete, paquete combinado, paquete simple, retardo, sistema lineal de ecuaciones, suma en campo.

Abstract

Multicast routing networks are established for transmission from a sending source node to a set of recipient sink nodes. However, these networks in their routing process can repeat packets by some links and, also, they can delay the transmission since they must create queues in the exit interfaces towards the destination nodes. The nodes that set them, work with the storage and forwarding policy, in such a way that the packets must be forwarded, in the best of cases, in the order in which they enter the router node. During the process of configuring the multicast routing network, several output links can be formed from the source node, leading to the arrival of multiple packets simultaneously to the intermediate routing nodes, generating these delays. The presence of Network Coding in conjunction with the multicast routing allows that within these intermediate nodes, as in the source node and the sinks, the linear combination of the incoming packets can be carried out. It generates an output coded packet of the same size as the incoming ones, increasing the use of bandwidth due to no waiting for in the output interface buffers. With Network Coding two problems with multicast routing must be solved. The first is to find the common multicast network of minimum-maximum flow that reaches the set of sinks. The second is the search for a solution scheme to a linear system of equations, for the delivery of the packets, sent simultaneously from the source node. However, finding the multicast network from a general communications network, and that can respond to packet coding, is not an easy task, given the arbitrary nature of the network. In the first place, in this thesis work, three algorithms are proposed that configure a communication multicast network, tending to have a solution for the reception of the minimum-maximum flow of simultaneously sent packets. The algorithms are based on the Ford-Fullkerson method with BFS and DFS, and on an exhaustive search algorithm for disjunct paths. Secondly, a method is proposed to find, from the source node of the multicast graph, the order of departure of the packets that make up the minimum-maximum flow, and that allow their decoding in the sink nodes.

Keywords: Algorithms, bandwidth, breadth search, depth search, path, augmented path, disjoint paths, finite field, packet coding, linear network coding, collision, linear combination of packets, cut, bottleneck, packet decoding , symmetric difference of sets, incoming links, outgoing links, router, vector space, maximum flow, communications graph, directed acyclic graph, directed labeled linear graph, multicast graph of minimum-maximum flow, adjacency matrix, minimum-maximum flow, nilpotent, encoder node, source node, sink node, packet order, packet, combined packet, simple packet, delay, linear system of equations, sum in field.

Contenido

Lista de Figuras

Lista de Tablas

Lista de Acrónimos

BFS	Breadth First Search.
BFS-PRI	BFS Eliminación de Primeros Caminos.
BFS-ML	BFS Eliminación de Caminos más largos.
DAG	Directed Acyclic Graph.
DFS	Depth First Search.
DFS-PRI	DFS Eliminación de Primeros Caminos.
DFS-ML	DFS Eliminación de Caminos más largos.
DLLG	Directed Labeled Line Graph.
IP	Internet Protocol.
LNC	Linear Network Coding.
MPH	Mínimum Cost Path Heuristic.
NC	Network Coding.
RLNC	Random Linear Network Coding.
XOR	Exclusive Or.

Lista de Símbolos

$\|a\|$	Longitud del vector a.
mod	Residuo de división entera.
⌈ ⌉	Función parte entera techo.
||	Operador de concatenación de vectores.
| ó :	Tal que.
:	Separador de límites de rango.
[ ]	Corchetes para encerrar elementos de un arreglo vector o matriz o un rango.
det	Determinante de una matriz cuadrada.
∧	Y-lógico.
∨	O-lógico.
~	No-lógico.
∀	Para todo.
∃	Existe un.
⇔	Si y solo sí.
⇒	Entonces.
∈	Pertenece a.
∉	No pertenece a.
∪	Unión de conjuntos.
∩	Intersección de conjuntos.
\	Diferencia de conjuntos.
Δ	Diferencia simétrica de conjuntos.
⊂	Subconjunto propio.

$\subseteq$	Subconjunto.
$\emptyset$	Conjunto vacío.
$\mapsto$	Asignación mediante una función.
$<$	Menor que.
$\leq$	Menor o igual.
$=$ ó $==$	Igualdad lógica.
$\neq$	Diferencia lógica.
$>$	Mayor que.
$\geq$	Mayor o igual.
$\preccurlyeq$	Subgrafo.
$\rightarrow$	Enlace entre dos nodos de un grafo.
$\rightsquigarrow$	Camino desde un nodo a otro.
Σ	Conjunto de símbolos del alfabeto de mnemónicos.
$\mathbb{Z}$	Conjunto de números enteros.
$\mathbb{F}_q$	Campo o cuerpo finito con q elementos.
$+$	Suma en $\mathbb{Z}$ o en $\mathbb{F}_q$.
$++$	Incrementar en uno.
$--$	Decrementar en uno.
$\sum$	Sumatoria de elementos en $\mathbb{Z}$ o en $\mathbb{F}_q$.
$\bigvee$	Operador $xorvect$ de conjuntos.
A^T	Transpuesta de la matriz A.
$log_b(q)$	Logaritmo en base b de q.
∞	Infinito.

Capítulo 1

Introducción

1.1 Network Coding y las Redes Multicast

El mecanismo de enrutamiento multicast [1] se ha utilizado para la transmisión de mensajes desde un computador origen hacia un subconjunto de computadores en la red. Este grupo de computadores conforman el conocido grupo multicast. En este tipo de redes, los datos se envían una sola vez, y lo reciben todos los destinos del grupo. Los enrutadores de la red se encargan de replicar los mensajes por las salidas necesarias para que alcancen todos los nodos del grupo. El grupo multicast es identificado por una dirección de grupo IP multicast. En especial, estas redes se han utilizado para transmisión de videoconferencia y teleconferencia a múltiples destinos en distintos puntos geográficos.

En los últimos años se ha aumentado en forma acelerada el crecimiento de usuarios de Internet [2] y, por ende, el crecimiento en el consumo de ancho de banda. La mayoría de las aplicaciones están enfocadas al tráfico de video y voz, sin despreciar las aplicaciones que transfieren datos de texto plano. Esto conlleva la implementación de métodos que mejoren la Calidad de Servicio [3], [4] a través de un adecuado manejo del ancho de banda y, consecuentemente, disminuyan valores como el retardo y el jitter.

Dado que las redes multicast requieren una alta demanda de ancho de banda, determinada por la transmisión y replicación de mensajes a través de los múltiples enlaces que alcanzan los computadores destinos, se ha propuesto la implementación de la técnica matemática de Network Coding [5] para la entrega simultánea de paquetes (mensajes) desde el nodo fuente (origen) hasta los nodos sumideros (destinos) de un grupo multicast. Esta técnica aplicada a las redes multicast permite entregar simultáneamente el flujo máximo común de paquetes entre el nodo fuente y los nodos sumideros, asumiendo una red acíclica y dirigida. El flujo máximo común corresponde al mínimo de los flujos máximos obtenidos, según el algoritmo de Ford-Fullkerson [6], entre el nodo fuente y cada uno de los nodos sumideros (grafos unicast de flujo máximo).

Con el apoyo de Network Coding, se cambia el paradigma de funcionamiento de los nodos de enrutamiento, pasando de ser, tradicionalmente, nodos de almacenamiento y reenvío a nodos de codificación. Estos nodos llevan a cabo la combinación lineal de los paquetes que pertenecen al campo $\mathbb{F}_q, q = 2^m$, transferidos por los enlaces entrantes, obteniendo un nuevo paquete que será reenviado por los enlaces de salida distintos a los entrantes, hasta alcanzar los nodos sumideros del grupo multicast. Los nodos sumideros actúan como nodos de decodificación para obtener los paquetes originales enviados desde el nodo fuente. Sobre la red multicast, se definirá un esquema de codificación con Network Coding, si y solo sí, se logra encontrar los paquetes originales en todos los nodos sumideros a través de la decodificación.

Este trabajo de tesis, propone un método para determinar un esquema de codificación con Network Coding, mediante la obtención de una secuencia ordenada de los paquetes por los enlaces de salida del nodo fuente de una red multicast de mínimo flujo máximo. El método se basa en determinar la independencia lineal de las combinaciones de paquetes que arriban a los nodos sumideros con el fin de hallar la solución que derive en los paquetes enviados desde el nodo fuente. Si no se logra hallar el ordenamiento de paquetes por las salidas del nodo fuente, el sistema no tendrá solución en los nodos sumideros, y estos no podrán determinar los paquetes originales enviados.

1.2 Enrutamiento Multicast para solución con Network Coding

Las redes de comunicaciones están formadas por múltiples enlaces y nodos de enrutamiento, constituyendo un modelo de grafo general de comunicaciones con los nodos fuente y sumideros de una red multicast definidos. A partir de estas redes, se debe obtener el grafo de enrutamiento multicast que permita transferir los paquetes desde el nodo fuente hasta el grupo de nodos sumideros en forma simultánea con el fin de lograr el máximo flujo común (o mínimo flujo máximo); así como permitir lograr el mínimo de nodos codificadores. El grafo multicast a obtener debe ser unisesión, dirigido, acíclico y con enlaces de capacidad igual a una unidad de información. La unidad de información establecida en este trabajo es un paquete de longitud l bits. El grafo multicast se considera unisesión, porque tiene un nodo fuente y un grupo de nodos sumideros receptores de la misma información.

Es importante obtener el menor número de nodos codificadores, ya que estos agregan tiempo adicional de procesamiento, producto de la codificación lineal que deben realizar sobre los paquees entrantes, para obtener el paquete a enviar por sus enlaces de salida.

La reducción del grafo general a un grafo multicast unisesión de mínimo flujo máximo es resuelta, en este trabajo de tesis, mediante tres algoritmos. Dos algoritmos utilizan el método de Ford-Fullkerson, los cuales sufren variaciones al momento de eliminar caminos de los grafos unicast que se forman entre el nodo fuente y algún sumidero que sobrepase el límite del mínimo flujo máximo. Estas variaciones están orientadas a eliminar los primeros caminos o los caminos más largos hasta igualar con el mínimo flujo máximo, el número de caminos en cada grafo unicast, antes de mezclarlos para obtener el grafo multicast unisesión de flujo máximo común. El tercer método, lleva a cabo una búsqueda exhaustiva para obtener los caminos más cortos, disyuntos y con menor número de sumideros entre el nodo fuente y los nodos sumideros.

El esquema de codificación propuesto en el trabajo, permite la obtención de los paquetes originados por el nodo fuente en los nodos sumideros. Se establece mediante un conjunto de restricciones definidas por un sistema lineal de ecuaciones soportadas por Network Coding, en las que los paquetes entrantes en los nodos sumideros, deben especificarse en función de los paquetes originales salientes en el nodo fuente. Además, se debe cumplir que el contenido de los paquetes entrantes debe ser linealmente independiente, para que se logre la solución del sistema en los nodos sumideros. La propuesta permite, como alternativa de solución, que se puedan establecer combinaciones lineales de los paquetes originales desde el mismo nodo fuente y no solo en los nodos codificadores interiores.

1.3 Esquema de la tesis

Este trabajo de tesis tiene dos logros importantes. En primer lugar, la reducción del grafo general de comunicaciones a un grafo multicast unisesión con un número de caminos disyuntos igual al mínimo flujo máximo, desde el nodo fuente hasta cada sumidero. Esta solución genera el menor número de nodos codificadores en promedio, y resuelve la mayor cantidad de grafos multicast de prueba sobre la base de Network Coding. En segundo lugar, la definición de un modelo de solución multicast para los grafos multicast unisesión de mínimo flujo máximo, que ordena los paquetes en los enlaces de salida desde el nodo fuente, de tal forma que los nodos sumideros reciban las combinaciones lineales necesarias que permitan obtener los paquetes originales. Para demostrar estos logros, la tesis está organizada como sigue:

El Capítulo 2 contiene los conceptos preliminares y básicos necesarios que se utilizarán dentro del marco de desarrollo del trabajo. Estos temas comprenden, dentro del álgebra lineal y la teoría de conjuntos, los tópicos de nilpotencia de matrices y sus propiedades básicas, la transposición de matrices, y la diferencia simétrica de conjuntos. También dentro del capítulo se presentan, desde la teoría de grafos, los conceptos de red general de comunicaciones, enrutamiento multicast y árboles de Steiner, junto con sus modelos matemáticos. Se presentan los conceptos acerca de Network Coding, su fundamentación matemática, ventajas, aplicaciones y limitaciones, y estado del arte. Además, se exponen los antecedentes de la problemática a resolver dentro del contexto de Network Coding. Se definen los conceptos básicos asociados con el cálculo del grafo de mínimo flujo máximo orientado a la solución con Network Coding. Estos conceptos comprenden: el grafo unicast de flujo máximo, la mezcla de grafos unicast, el modelo de enrutamiento multicast para Network Coding, y el método para calcular todas las rutas desde un nodo fuente hasta un sumidero en un grafo. Otros conceptos básicos están dentro del área de teoría de campos y espacios vectoriales, tales como la definición de símbolos, códigos y campos, y espacios vectoriales y paquetes. Por último, se presentan los dos grandes problemas a resolver, dentro del marco de desarrollo de este trabajo; así como los objetivos, general y específicos, que se persiguen.

En el Capítulo 3, se explica, en la primera parte, el concepto de redes de flujo, el cual comprende las temáticas relacionadas con su modelamiento matemático en grafos, sus propiedades y características, y el máximo flujo. Se introduce el concepto de corte, la capacidad del corte y el mínimo corte. Se expone el problema del flujo máximo que abarca los conceptos de red residual, camino aumentado y enlace saturado, y el algoritmo de flujo máximo propuesto por Ford-Fullkerson. Se explica el problema del mínimo corte, el flujo dentro del corte, y sus propiedades. Por último, se presenta el teorema del Flujo Máximo – Corte Mínimo, que relaciona ambos conceptos, que son relevantes para hallar un grafo de mínimo flujo máximo a partir de un grafo general de comunicaciones. En la segunda parte, el Capítulo 3 explica el método para hallar el grafo multicast dirigido, acíclico y unisesión de mínimo flujo máximo a partir de un grafo de general de comunicaciones, basado en el método de Ford-Fullkerson. Con el algoritmo de Ford-Fullkerson, se proponen las versiones de búsqueda en anchura y búsqueda en profundidad para obtener cada grafo unicast de flujo máximo desde el nodo fuente hasta cada nodo sumidero. De los grafos unicast obtenidos, se calcula el mínimo flujo máximo, y se eliminan los primeros caminos o los caminos más largos de los grafos que sobrepasan el mínimo flujo máximo hasta igualarlos todos al mínimo flujo máximo. Por último, se mezclan los grafos de mínimo flujo máximo para obtener el grafo multicast objetivo, deduciendo de este, los nodos codificadores.

El Capítulo 4 presenta una solución mejorada al problema de reducir el grafo general de comunicaciones al grafo multicast de mínimo flujo máximo unisesión, dirigido y acíclico. Este método se basa en el cálculo de todos los caminos desde el nodo fuente hasta cada sumidero. A partir del conjunto de caminos resultantes por cada sumidero, se obtienen los caminos disyuntos más cortos entre el nodo fuente y cada sumidero, conformando un grafo unicast de flujo máximo. De estos, se deduce el mínimo flujo máximo común. Posteriormente, se obtiene la unificación de la cantidad de caminos que conforman el mínimo flujo máximo en los grafos que lo sobrepasan. Se seleccionan, en lo posible, los caminos de estos grafos que se sobreponen a caminos de grafos unicast que si cumplen con el mínimo flujo máximo. Esto permite el transporte de un paquete por un camino común con dirección a más de un sumidero, y disminuye la aparición de nodos codificadores o caminos que conducirán tráfico de paquetes inconsistentes que no se podrán decodificar en los nodos sumideros. Luego, los grafos unicast, ya igualados en el mínimo flujo máximo, son mezclados para obtener el grafo multicast de mínimo flujo máximo. Sin embargo, el método puede arrojar soluciones con enlaces entrantes en los sumideros, que entregan paquetes con posibilidad de anularse, y llevar a un sistema sin solución. El método de caminos disyuntos, aplica un algoritmo de corrección basado en la diferencia simétrica de conjuntos (considerando los nodos adyacentes al nodo fuente como conjuntos), para eliminar enlaces que finalizan en nodos codificadores, y así cambiarlos a nodos de almacenamiento y reenvío, minimizando el problema anterior.

El Capítulo 5 muestra el método mediante el cual se determina, en los nodos sumideros de un grafo de mínimo flujo máximo, la posibilidad de hallar los paquetes enviados por el nodo fuente en forma simultánea, a través de la solución de un sistema lineal de ecuaciones sobre la base de Network Coding. También se determina el orden en que deben emerger los paquetes por los enlaces de salida desde el nodo fuente. En este capítulo, se redefine el concepto de tasa máxima de transmisión, se define el modelo de codificación de los mensajes, se plantea el problema de solución del sistema lineal de ecuaciones sobre Network Coding para una red multicast unisesión, se presentan los componentes del modelo de red para construir la solución y, por último, se plantean las bases y el esquema de restricciones que dará solución al sistema. Al final, se muestran ejemplos de aplicación del modelo con grafos multicast unisesión de mínimo flujo máximo, y un resumen de las características de la solución planteada; además de una comparación con el método polinomial de solución de un sistema multicast soportado por Network Coding.

En el Capítulo 6, se exponen las valoraciones de los métodos propuestos de reducción del grafo de comunicaciones a un grafo multicast de mínimo flujo máximo, utilizando las variables de número de soluciones con paquetes simples, número de soluciones con paquetes combinados, y número promedio de nodos codificadores resultantes en los grafos multicast. También determina cuántos métodos solucionan de la forma más óptima un grafo multicast, y cuántos métodos son únicos solucionando un grafo multicast. Por último, se revisan los métodos de reducción a partir del tiempo de ejecución de cada algoritmo.

El Capítulo 7 presenta las principales conclusiones derivadas de este trabajo, así como los posibles tópicos de investigación futuros. Es importante destacar, que los resultados de esta tesis, se pueden extender a una red multicast multisesión con diferentes mínimos flujos máximos entre los nodos fuentes y sus distintos grupos multicast.

Capítulo 2

Preliminares

2.1 Conceptos Básicos de Algebra Lineal y Teoría de Conjuntos

2.1.1 Matrices nilpotentes y propiedades

Una matriz A diferente de cero es nilpotente, si existe un número entero k tal que $A^k = 0$. El índice de nilpotencia se define como el entero más pequeño para el que $A^k = 0$ [7].

Lema 2.1: Si A es nilpotente, entonces $\det A = 0$.

Demostración:
Si A es nilpotente, entonces existe un $k > 0$ tal que $A^k = 0$, luego $\det(A^k) = 0$ y, además, $\det(A^k) = \det \underbrace{(AA \dots A)}_{k-veces}$=0, luego $\underbrace{\det(A)\det(A) \dots \det(A)}_{k-veces}$=0, de donde $(\det A)^k = 0$ y, por último, $\det A = 0$. ■

Lema 2.2: Si A es una matriz triangular superior con ceros en la diagonal principal, entonces A es nilpotente.

Demostración:
Por inducción matemática [8] sobre k en una matriz cuadrada A de tamaño $n \times n$ se tiene:
Base:
Si $n = 2$, la matriz triangular superior A con ceros en la diagonal principal es de tamaño 2×2, luego $A^2 = 0$, por lo tanto $k = 2$.
Paso Inductivo:
Se asume verdadero para un índice de nilpotencia entero k. La matriz triangular superior A con ceros en la diagonal principal de tamaño $n \times n$, se puede expresar como $A = \begin{bmatrix} A_{11} & A_{12} \\ 0 & 0 \end{bmatrix}$, donde el bloque matricial A_{11} es de tamaño $(n-1) \times (n-1)$, A_{12} es de tamaño $(n-1) \times 1$, A_{21} es de tamaño $1 \times (n-1)$ y A_{22} de tamaño 1×1. La matriz A_{11} es triangular superior con ceros en la diagonal principal, las matrices A_{21} y A_{22} son 0 y la matriz A_{12} toma cualquier valor. Se asume que existe un entero k tal que $A^k = 0$; es decir, $\begin{bmatrix} A_{11}^k & A_{11}^{k-1}A_{12} \\ 0 & 0 \end{bmatrix} = 0$, por tanto $A_{11}^k = 0$ y A_{11} es nilpotente con k como índice de nilpotencia.
Para $k+1$, se demuestra que $A^{k+1} = \begin{bmatrix} A_{11}^{k+1} & A_{11}^k A_{12} \\ 0 & 0 \end{bmatrix} = \begin{bmatrix} A_{11}^k A_{11} & A_{11}^k A_{12} \\ 0 & 0 \end{bmatrix} = \begin{bmatrix} 0 & 0 \\ 0 & 0 \end{bmatrix} = 0$, ya que $A_{11}^k = 0$, según la hipótesis. ■

2.1.2 Transpuesta de una Matriz

Sea $A = (A_{ij})$ una matriz de tamaño $n \times m$. La matriz transpuesta de A, que se denota por A^T, es la matriz de tamaño $m \times n$, obtenida al intercambiar las filas por las columnas de A [7], [9]. Se puede describir la transpuesta como $A^T = (A_{ji})$.

Se denota $A^T(j)$ como el vector correspondiente a la j-ésima fila de la matriz A^T, que corresponde con la j-ésima columna de A.

2.1.3 Diferencia Simétrica de Conjuntos

La diferencia simétrica de dos conjuntos [10]–[12] A y B, se define como

$$A\Delta B = (A \backslash B) \cup (B \backslash A) = (A \cup B) \backslash (A \cap B) \tag{2.1}$$

Es decir, la diferencia simétrica de dos conjuntos genera un conjunto cuyos elementos pertenecen a uno de los conjuntos, sin pertenecer a ambos.

2.2 Red de Comunicaciones

Para una sesión de comunicaciones unidifusión [13], la red de flujo [14], se modela con un grafo de enlaces dirigidos, acíclico y sin pérdidas (completamente confiables) $G = (V, E)$, donde V representa el conjunto de nodos (enrutadores) y E es el conjunto de enlaces dirigidos que unen los nodos. Sean $i, j \in \mathbb{N}$, se establece que $\forall e \in E$, se puede representar como el par ordenado $e = (i, j)$ ó $e = ij$ (en este trabajo, utilizaremos ambas notaciones), donde $i, j \in V$, están ordenados topológicamente [14]–[16], es decir $i < j$ (o i "precede a" j) y el flujo de información va desde el nodo i al nodo j, no considerando retardos en la transmisión. El nodo origen y destino de un enlace $e = (i, j)$, se denota por $o(e) = i$ y $d(e) = j$, respectivamente. Dentro del grafo G se diferencian el nodo fuente s y el nodo sumidero t. Cada nodo $v \in V \setminus \{s, t\}$ se encuentra en algún camino desde s hasta t, logrando establecer un grafo conectado, y $|E| \geq |V| - 1$.

Se define de [17], $\Gamma_I(i)$ como el conjunto de enlaces que ingresan al nodo $i \in V$ y $\Gamma_O(i)$ como el conjunto de enlaces que se originan en i. De manera formal, se escribe

$$\Gamma_I(i) = \{e \in E |\, d(e) = i\} \tag{2.2}$$

$$\Gamma_O(i) = \{e \in E |\, o(e) = i\} \tag{2.3}$$

Además, se define $\delta_I(i)$ como el grado o número de enlaces que ingresan a i, mientras que $\delta_O(i)$ es el grado o número de enlaces que emergen de i. Formalmente, $\delta_I(i) = |\Gamma_I(i)|$ y $\delta_O(i) = |\Gamma_O(i)|$. El nodo s no tiene enlaces de entrada y el nodo t no tiene enlaces de salida; es decir, $\Gamma_I(s) = \emptyset$ y $\Gamma_O(t) = \emptyset$.

A cada enlace $e \in E$, se le asocia un $C(e) \in \mathbb{Z}^+$, llamado la capacidad de e. Un nodo i puede enviar información a través de un enlace $e = (i, j)$ a una tasa máxima de $C(e)$ bits por unidad de tiempo. Además, $C(i, j) = 0$ si $(i, j) \notin E$. El flujo de bits (llamado proceso aleatorio en [17],[18]) transmitido a través del enlace e se denota por f_e o $f(i, j)$. Además, al nodo i ingresan los flujos $f_{e'}$ que transitan por cada $e' \in \Gamma_I(i)$.

2.3 Enrutamiento Multicast

2.3.1 Modelo de grafo multicast unisesión

Una red como Internet está constituida por un conjunto de enlaces y enrutadores, los cuales no son todos necesarios al momento de establecer los caminos de enrutamiento desde un nodo fuente hacia al conjunto de nodos sumideros en una sesión de comunicación multicast.

Los sistemas de redes de comunicaciones se diseñan para entregar la información desde nodos fuentes a nodos destinatarios. La forma tradicional de entrega de datos emplea caminos para la conexión unidifusión y árboles para las conexiones multicast [19]. Desde el punto de vista de [20], multicasting es la capacidad de una red de comunicaciones para generar un mensaje simple desde una aplicación, y entregar copias del mensaje a múltiples receptores en diferentes localizaciones. Uno de los principales retos, es disminuir el uso de los recursos de red.

Deering propuso en 1990, el IP multicast, como una extensión al modelo de servicio unicast, con el fin de lograr una comunicación eficiente multipunto[21]. Aunque el envío de información a un grupo, se puede realizar a través del envío de diferentes mensajes unicast a cada uno de los hosts destinos, es preferible utilizar multicast para realizar esta tarea. La principal ventaja de usar multicast es la disminución de la carga en la red.

Por otro lado, desde el punto de vista formal de grafos, una red multicast [22] es un grafo dirigido $G' = (V', E')$, donde los elementos de V' son denominados nodos y los elementos de E' son llamados enlaces de un nodo a otro. G' contiene un nodo fuente $s \in V'$, que puede transmitir la misma información a un conjunto de nodos destinatarios (sumideros) $T \subset V'$, donde $s \notin T$. El nodo fuente tiene un conjunto de mensajes de un alfabeto fijo y cada destino intentará recuperar todos los mensajes. Los elementos de $V'\backslash T\backslash\{s\}$ son nodos interiores. Para una red multicast, se requiere que $\Gamma_I(i) \geq 1, \forall i \in V'\backslash\{s\}$.

El principal problema a resolver para el enrutamiento multicast general, consiste en determinar la red que representa una sesión para la entrega confiable de los paquetes originados en un nodo fuente s a un conjunto $T \subset V'\backslash\{s\}$ de nodos sumideros atravesando un grupo de nodos intermedios.

Según [23], en general, una red de sesión multicast corresponde a $G' = (V', E') \preccurlyeq G = (V, E)$, donde G' es un subgrafo de G, $V' \subseteq V$ y $E' \subseteq E$. De igual forma, en el subgrafo G' los enlaces $e' \in E'$ tienen asociados sus capacidades $C(e')$. De esta definición, se infiere que $|V'| \leq |V|$ y que $|E'| \leq |E|$. En la red de sesión multicast, un nodo $s \in V'$ corresponde con la fuente de los mensajes que se enviarán a los nodos sumideros $T \subset V'\backslash\{s\}$.

2.3.2 Arbol de Steiner

Con el fin de llevar a cabo la comunicación multicast, los nodos del grupo multicast deben estar interconectados a través de un árbol [20]. Por tanto, el problema de enrutamiento multicast en una red de comunicaciones se reduce a encontrar un árbol A en un grafo de comunicaciones G, tal que A abarque todos los nodos en el grupo multicast. Este árbol es llamado un árbol multicast, cuya obtención se convierte en un problema de optimización de enrutamiento multicast que se resuelve mediante la consecución del árbol de *Steiner* [24]–[28]. Sin embargo, se ha demostrado que

encontrar el árbol de *Steiner* óptimo, que proporcione el máximo rendimiento, es un problema NP-completo [23], [27].

Para definir el problema de optimización en redes multicast, se define el problema del árbol de *Steiner* en los siguientes términos [20], [29]: Sea $G = (V, E)$ un grafo no dirigido con pesos w_e no negativos en cada enlace $e \in E$, y un conjunto de nodos $M \subseteq V$, los cuales pertenecen al grupo multicast, se debe encontrar un árbol $A = (V_A, E_A)$ que abarque M, y tal que su peso, W_A, calculado en (2.4), sea mínimo.

$$W_A = \sum_{e \in E_A} w_e \tag{2.4}$$

Ejemplo 2.1: La Figura 2.1 muestra un grafo de comunicaciones G de 12 nodos (enrutadores) y 19 enlaces. En este grafo, se destaca un árbol de *Steiner* con los nodos y enlaces resaltados. El árbol de *Steiner* conecta el grupo multicast conformado por los nodos $M = \{CA1, TX, IL, NY\}$. En el ejemplo, se asume que el peso de cada enlace es 1, por lo tanto, el peso del árbol de *Steiner* es 5. Los nodos $CA2$ y PA no pertenecen al grupo multicast, pero si son parte del árbol de *Steiner*, y se les denomina nodos *Steiner*. El nodo fuente s está etiquetado con $CA1$ y los tres nodos sumideros son $T = \{TX, IL, NY\}$. En general, el grafo multicast $G' = (V', E')$, donde $V' = \{CA1, CA2, PA, TX, IL, NY\}$ y E' consta de 5 enlaces, los cuales están resaltados.

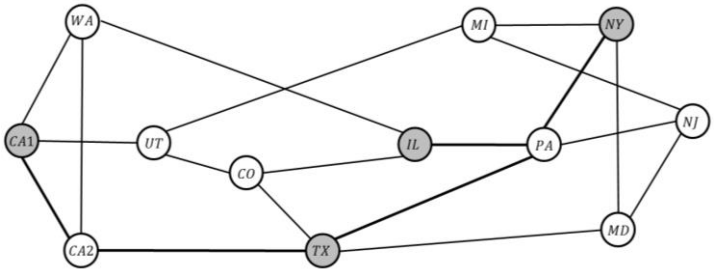


Figura 2.1 Ejemplo del árbol de *Steiner* [20]

2.3.3 Mecanismo de enrutamiento Multicast

En los sistemas unidifusión, se utilizan mecanismos de enrutamiento para llevar a cabo la entrega de información. Los mecanismos de enrutamiento tradicionales, permiten que los nodos intermedios (enrutadores) ejecuten las acciones de almacenamiento y reenvío, de tal manera que cada mensaje de salida es una copia de un mensaje recibido previamente en un enlace de entrada al nodo. En los sistemas multicast IP tradicionales [30], además que los nodos intermedios reciben los paquetes por los enlaces de entrada, pueden duplicarlos y reenviarlos a varios enlaces de salida de acuerdo con el método de enrutamiento seleccionado, independientemente de la correlación del contenido entre los datos para los distintos destinos.

Las rutas multicast para diferentes destinos, se pueden intersectar en algunos nodos y compartir enlaces comunes. En este caso, es inevitable el uso de políticas de encolamiento para los diferentes paquetes de datos, lo cual resultará en la reducción de la eficiencia en la transmisión, y hace este mecanismo proclive a la congestión de tráfico. Por lo tanto, no es posible lograr el máximo flujo simultáneamente para todos los nodos sumideros en T.

2.4 Network Coding

2.4.1 Conceptos y generalidades

Por su lado, Network Coding [5], [17], [31] permite que cada enrutador que actúa como nodo en la red pueda llevar a cabo una mezcla de los mensajes a través del cálculo de su combinación lineal, no solo en los nodos fuente y sumideros, sino en los nodos intermedios. La Figura 2.2 muestra un nodo (enrutador) al que ingresan tres flujos de mensajes y se obtienen dos flujos salientes que son una combinación lineal de los entrantes. En general, el flujo de bits f_e transmitido a través del enlace $e = (i,j) \in \Gamma_O(i)$ será una función de todos los flujos $f_{e'}$, donde $e' \in \Gamma_I(i)$. Cuando se utiliza Network Coding en las transmisiones, especialmente multicast, los mensajes pueden ser reenviados, mezclados (codificados) en los nodos intermedios, y decodificados en los nodos sumideros para obtener los mensajes originales [13]. Por tanto, la teoría de Network Coding muestra que la transmisión de información en forma multicast puede ser mejorada, superando el paradigma de enrutamiento y replicación de datos.

Según [32], "*Network Coding significa que los bits en los flujos de información no tienen que ser entregados como una mercancía; estos pueden ser mezclados como se quiera, siempre y cuando los computadores receptores hayan recibido suficiente evidencia o pistas para reconstruir los paquetes originales del computador emisor*."

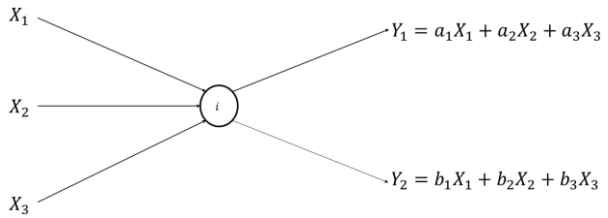


Figura 2.2 Cálculo de mensajes de salida a partir de mensajes de entrada

Tomando la Figura 2.3 [13], [19], se muestra una red constituida por enlaces dirigidos que tienen la misma capacidad de B bps. En esta red, los computadores fuente s_1 y s_2 envían un flujo de mensajes, sin tener nodos enrutadores intermedios, a los computadores receptores t_2 y t_1, respectivamente. A su vez, el computador fuente s_1 intenta enviar mensajes al computador t_1, y el computador s_2 intenta enviar mensajes al computador t_2. Se puede observar que los mensajes salientes de s_1 pueden llegar a t_1 solamente a través del camino $s_1 \rightarrow 1 \rightarrow 3 \rightarrow 4 \rightarrow 6 \rightarrow t_1$, y los mensajes salientes de s_2 pueden llegar a t_2 solamente a través del camino $s_2 \rightarrow 2 \rightarrow 3 \rightarrow 4 \rightarrow 5 \rightarrow t_2$. Claramente se nota que el enlace (3,4) es compartido y actúa constituyendo un cuello de botella para la transmisión simultánea.

Si los nodos, que conforman la red de la Figura 2.3, solo enrutan la información que reciben, el enlace (3,4) debe ser usado en forma compartida entre las dos sesiones unicast, permitiendo solo el siguiente conjunto de tasas de transmisión $\{(r_1, r_2) | 0 \leq r_1, 0 \leq r_2, r_1 + r_2 \leq B\}$, el cual es interpretado gráficamente en la Figura 2.4(a) [33]. No obstante, si los nodos de la red pueden ejecutar la codificación de red, específicamente en el nodo 3 del enlace compartido (3,4), luego ambas sesiones pueden comunicarse confiablemente a la tasa B, permitiendo el conjunto de tasas de transmisión $\{(r_1, r_2) | 0 \leq r_1 \leq B, 0 \leq r_2 \leq B\}$, como se muestra en la Figura 2.4(b) [33]. Para ambas sesiones unicast, se logra la tasa B a través de la mezcla de los flujos de información en el

nodo 3 utilizando una operación XOR, y luego en los nodos 5 y 6, se logra obtener los flujos originales aplicando nuevamente la operación XOR. En la Figura 2.3, los flujos de información original se transportan en los enlaces de rectas quebradas, mientras que el flujo mezclado se transporta en los enlaces de rectas continuas.

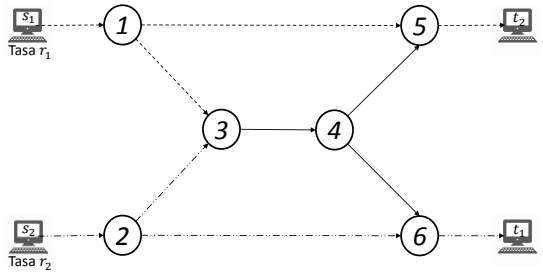
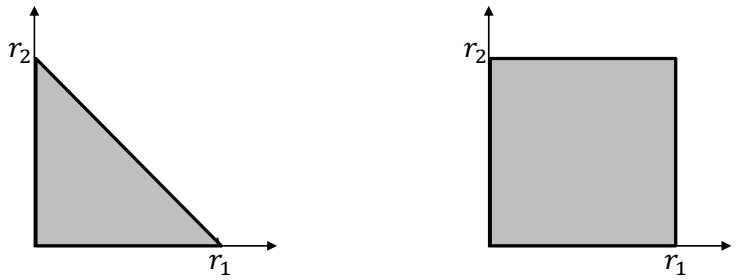


Figura 2.3 Dos sesiones unicast en contienda por el enlace (3,4)[13]

No es posible, para ninguna de las sesiones unicast, establecer un flujo de comunicación con una tasa mayor que B, ya que los enlaces en la red están limitados por esta capacidad para cualquier sesión entre un emisor y los receptores. De lo anterior, la región de la Figura 2.4(b) comprende todas las tasas de flujo alcanzables, la cual es llamada la región de capacidad para estas sesiones [13], [33].



(a) (b)

Figura 2.4 Rangos de tasas de transferencia de datos alcanzables para (a) enrutamiento tradicional y (b) enrutamiento con NC[13]

Según Fragouli *et al.* [34], las redes de comunicaciones actuales comparten el mismo principio fundamental de operación. Independientemente de que se envíen paquetes sobre Internet, o señales en un red telefónica, la información es transportada de la misma forma como los vehículos comparten una autopista. Es decir, los flujos de datos independientes pueden compartir los recursos de la red, pero la información en sí está separada.

Actualmente y, en forma tradicional, las redes de comunicaciones tratan los paquetes de datos como una unidad inmodificable; es decir, los paquetes de datos que se producen en la fuente se enrutan a través de la red hasta que alcanzan un destino. En este proceso, cada paquete se mantiene intacto.

El paradigma de Network Coding (NC) propuesto inicialmente en [5], es un campo reciente en la teoría de información que rompe con el supuesto anterior. Este propone, que en vez de simplemente reenviar datos, los nodos pueden combinar varios paquetes de entrada en uno o varios paquetes salientes, siendo innecesario entregar a un nodo sumidero los paquetes reales producidos por el nodo fuente. Los nodos sumideros estarán en capacidad de obtener los paquetes originales enviados

a través de un proceso de decodificación de los paquetes recibidos. Un ejemplo sencillo en un contexto de redes inalámbricas, es una topología de tres nodos [35], [36], como se muestra en la Figura 2.5, donde el nodo 1 intercambia paquetes a través del nodo de reenvío 3 (estación inalámbrica de base) con el nodo 2. Se asume que la transmisión llevada a cabo es "semi-dúplex"; es decir, que durante cualquier intervalo dado, un nodo puede transmitir o recibir, pero no puede transmitir y recibir al mismo tiempo. Durante un período de tiempo, el nodo de reenvío (3) puede recibir transmisiones desde cualquier estación, pero no de ambas. Cuando el nodo de reenvío realiza la difusión de los paquetes, ambos nodos emisores-receptores (1 y 2) reciben el mensaje. Si se quiere enviar un paquete por cada estación (un paquete X_1 desde 1 a 2 y un paquete X_2 desde 2 a 3), se pueden establecer los siguientes escenarios:

Según la Figura 2.5(a) [35], [36], la solución sin Network Coding o con enrutamiento solamente, lograría el envío y entrega de los dos paquetes objetivos de la comunicación en cuatro espacios de tiempo: (t_1) el envío de X_1 desde el nodo 1 al nodo 3; (t_2) la difusión de X_1; (t_3) el envío de X_2 desde el nodo 2 al nodo 3; y (t_4) la difusión de X_2.

Según la Figura 2.5(b) [35], [36], la solución con Network Coding lograría la transmisión de los dos paquetes en solo tres espacios de tiempo: (t_1) el envío de X_1 desde el nodo 1 al nodo 3; (t_2) el envío de X_2 desde el nodo 2 al nodo 3; y (t_3) la difusión de $X_1 + X_2$ desde 3 a cada uno de los nodos emisores-receptores. Dado que el nodo 1 ya conoce a X_1, éste puede recuperar X_2 mediante el cálculo de $X_1 + (X_1 + X_2)$ y de igual manera, el nodo 2 puede recuperar a X_1 porque conoce a X_2. Se puede observar que a pesar del costo de la operación de codificación en el nodo interno de la red, y las operaciones de decodificación en los nodos destinos, el rendimiento mejora más allá de lo que se puede lograr a través del enrutamiento solamente. El rendimiento en el reenvío tradicional sería de $1/2$, y con la aplicación de Network Coding sería de $2/3$ (en el mejor de los casos, cuando X_1 y X_2 salgan al mismo tiempo de sus puntos de partida y lleguen al nodo 3, este realizará la codificación y reenviará el paquete codificado por cada uno de sus puertos de salida al mismo tiempo, realizando las dos actividades en los tiempos t_1 y t_2, derivando en un rendimiento de 1).

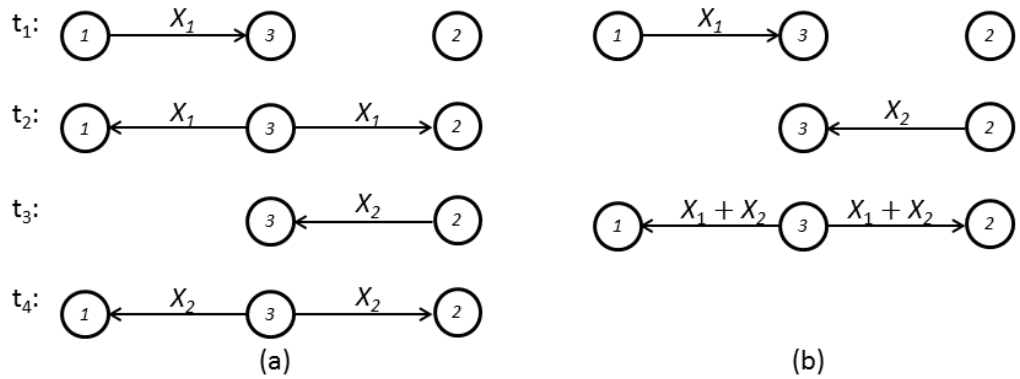


Figura 2.5 Intercambio de mensajes entre dos dispositivos inalámbricos: (a) Sin NC, (b) Con NC

2.4.2 Ventajas y Aplicaciones

Además de permitir una mejora en el rendimiento, Network Coding ofrece otros beneficios para la red multicast, tales como el balanceo de carga y el ahorro en la utilización del ancho de banda [37], [38] [30], [39]. En el mismo contexto de las redes de comunicaciones, Chou and Wu [13]

expusieron un importante trabajo sobre Network Coding y sus aplicaciones en redes de comunicaciones alámbricas e inalámbricas. Ellos presentaron varios aspectos tales como la sincronización, variación del retardo, y pérdida de datos.

En general, Network Coding ha emergido como un nuevo paradigma que ha influenciado distintas áreas de la informática [36], [40], tales como la ciencia de la computación [41], [42], almacenamiento de datos distribuidos [43], [44], seguridad de información [45]–[47], teoría de información [48], [49], la teoría de código [33], [50], [51], teoría de optimización [52], [53], entrega de contenidos en redes P2P reduciendo el retardo en la entrega de la información [54]–[57], redes de tolerancia a retardos (DTN) [58], [59], las comunicaciones inalámbricas/satelitales [35], [60], [61], redes ópticas [62]–[65] y teoría de switches [66]–[68].

2.4.3 Soluciones Multicast con Network Coding y sus limitantes

El trabajo inicial teórico en Network Coding desarrollado por Ahlswede *et al.* [5], fue modelado con un grafo dirigido $G = (V, E)$ con enlaces sin ruido y con capacidades de un símbolo de un campo $\mathbb{F}_q$ por unidad de tiempo, mostrando que un nodo fuente $s \in V$ puede enviar la misma información a un conjunto de nodos sumideros $T = \{t_1, t_2, \dots, t_d\} \subset V$, donde $s \notin T$, a una tasa de flujo de datos máxima. Ahlswede *et al.* demostraron que el problema de una sesión multicast se convierte en una subutilización de los recursos al restringir la utilización de los nodos de la red a la ejecución de solamente la función de enrutamiento. También demostraron en su trabajo, que la capacidad multicast $r(s, T)$, que es definida como la máxima tasa a la que un nodo fuente puede transmitir la información común a un conjunto de nodos sumideros, está determinada por el menor valor de los máximos flujos desde el nodo fuente s a cada uno de los nodos sumideros $t \in T$; es decir $r = min_{t \in T} MaxFlujo(s, t)$.

El resultado anterior es conocido como el teorema del *flujo de información*, que puede ser considerado como una generalización del clásico teorema de máx-flujo-mín-corte para *flujo de productos* [69], el cual fue probado de forma independiente por Elías *et al.*[70] y Ford y Fullkerson [6], [71] en 1956. Con el uso de Network Coding, se demostró el logro en el aumento del rendimiento, lo cual es una ventaja sobre los mecanismos de enrutamiento clásicos. De igual forma, demostraron que mientras no se pueda lograr la capacidad multicast por mecanismos de enrutamiento, esta se puede alcanzar a través de Network Coding.

De forma general, existen dos tipos de algoritmos de flujo máximo[30], [72]. Los algoritmos basados en caminos aumentados, tal como Ford-Fullkerson, Edmonds-Karp [73], [74] y Dinic [75], [76], que mantienen la conservación del flujo para cada nodo, excepto para el fuente y el destinatario. Esto significa que en cada nodo intermedio, los valores de los flujos de entrada y salida son iguales entre sí. Para estos casos, el valor del flujo se incrementa gradualmente hasta que alcanza el límite máximo.

Por otro lado, existen los algoritmos basados en el método del preflujo-empuje. Contrario a los primeros, estos algoritmos no garantizan la conservación del flujo. Un preflujo es similar a un flujo, excepto que la cantidad que fluye hacia un nodo puede exceder la cantidad que fluye desde el nodo [77], y de esta forma no se garantiza el principio de conservación. El valor del flujo en cada enlace es ajustado iterativamente [30]. Cuando todos los nodos intermedios satisfacen el principio

de conservación del flujo, ya se habrá determinado el flujo máximo. Entre las propuestas algorítmicas que trabajan con el método del preflujo-empuje, está el método sugerido por Karzanov [78], quien mejoró el algoritmo de Dinic en una red de capas. Otros algoritmos que también trabajan aplicando el método preflujo-empuje, son las presentadas por Golberg y Golberg-Tarjan [72], [77], [79], [80].

Para cualquier caso, ambos tipos de algoritmos se pueden implementar en un ambiente distribuido. Sin embargo, es más fácil en los algoritmos del primer tipo controlar el valor del flujo, ya que estos garantizan su conservación.

En [31] se demostró que es suficiente que la función de codificación en los nodos interiores sea lineal, para lograr la máxima capacidad multicast, definiendo así la Codificación de Red Lineal o *Linear Network Coding* (LNC). LNC considera los mensajes como vectores constituidos por elementos de un campo finito $\mathbb{F}_q$ [81], el cual puede ser representado por un vector binario de longitud $log_2(q)$ bits [19], y la función de codificación que genera un mensaje emergente de un nodo, es una simple combinación lineal de los mensajes entrantes al nodo en el espacio vectorial definido sobre $\mathbb{F}_q$. De igual manera, las decodificaciones en los sumideros pueden ser llevadas a cabo a través de operaciones lineales sobre la información entrante a los nodos.

Koetter y Médard [17], [82] mostraron que las soluciones lineales existen para redes multicast solucionables al interior de un alfabeto de algún campo finito, cuyo tamaño es una potencia de dos, y que es tan grande como $r|T|$. Además, demostraron cómo encontrar los coeficientes de la codificación lineal y las funciones de decodificación mediante la búsqueda de los valores de las variables de un polinomio no nulo. Sin embargo, para la solución del problema multicast, la estructura propuesta por ellos produce un algoritmo de tiempo polinomial. Ho *et al.* [83] propusieron la estructura de la Codificación de Red Lineal Aleatoria o *Random Linear Network Coding* (RLNC), la cual orienta la investigación de codificación de red de la teoría a la aplicación práctica y, además, muestran que la capacidad de una sesión multicast es alcanzable a través de LNC. En [84]–[86], se demostró cómo encontrar los coeficientes de codificación y decodificación lineal para una sesión multicast, a través de un algoritmo que se ejecuta en un tiempo polinomial en un campo finito de tamaño máximo $|T|$. De igual forma, en [87], fue demostrado este mismo tamaño máximo para el campo finito que comprende los coeficientes de codificación y decodificación multicast. Ellos ofrecieron una visión combinatoria para reformular las condiciones algebraicas propuestas en [83].

El trabajo expuesto en [88] demostró que para lograr una solución lineal, algunas redes multicast solucionables asintóticamente requieren alfabetos de campos finitos que sean al menos del tamaño $2\sqrt{|T|}$. La solubilidad es probada sobre una red construida especialmente, y limita su tamaño de alfabeto a una solución lineal. Además, la red que ellos utilizan no es solucionable sobre el campo binario. De igual forma, Rasala Lehman y Lehman [89] configuraron una red multicast similar a la de [35], y llegaron al mismo resultado. También, ellos determinaron los tipos de complejidades de diferentes problemas de codificación de red lineal escalar.

A través de la formulación del problema, utilizando la programación lineal, se obtuvieron aplicaciones en redes inalámbricas utilizando Network Coding para la solución multicasting de

optimización de energía [90]. En [91], se han mostrado los resultados de las investigaciones para la aplicación de Network Coding en redes con ciclos, siendo de los primeros en realizar este tipo de estudios, dado que en la mayoría de los trabajos se parte del supuesto que la red de comunicaciones es acíclica.

2.4.4 Antecedentes al problema planteado

De acuerdo con [92], [93], cuando se utiliza Network Coding en redes multicast, se deben considerar dos pasos: el primero, encontrar los caminos de transmisión propios desde el nodo fuente hasta los múltiples sumideros, de tal forma que constituya una red multicast de mínimo flujo máximo común que se pueda resolver bajo un esquema de codificación; y el segundo, determinar el esquema de codificación adecuado. En cuanto al último problema, se han presentado varias propuestas tales como [17], [31], [85], [86], [94]. La propuesta de RLNC [94], se caracteriza por ser una implementación distribuida de LNC, con una baja complejidad comparada con un algoritmo heurístico en [17], un algoritmo exponencial en [31], y un algoritmo exponencial en [85], [86]. Este método al ser comparado con otros métodos, tiene la más baja complejidad y puede ser utilizado en sistemas de redes reales.

El primer problema, sin embargo, no ha sido tratado tan ampliamente ni de manera profunda. El establecimiento del enrutamiento es una precondición para llevar a cabo la transmisión multicast en una red. La estructura del grafo de enrutamiento multicast es la principal condición para que se pueda dar solución a un problema de LNC. Los algoritmos actuales de enrutamiento multicast están fundamentados en la capa de red o capa IP, a través de la configuración de los árboles de *Steiner* [95]. Sin embargo, cuando se utiliza Network Coding en enrutamiento multicast, este problema es mucho más simple. En este caso, uno o más nodos intermedios en el grafo de red pueden codificar los paquetes de datos de múltiples flujos de información, que ingresan a través de diferentes enlaces, en un único paquete de datos y transmitirlo simultáneamente sin encolamiento a nodos adyacentes. Si no hay pérdida de información, los flujos de datos combinados pueden ser decodificados, obteniendo los paquetes originales en los nodos sumideros [30].

Un trabajo en esta línea, se presenta en [93], en el cual se propone un método para encontrar los caminos de transmisión propios desde el nodo fuente hasta los múltiples sumideros en una red multicast, mientras se usa Network Coding. El flujo máximo entre el nodo fuente y los nodos sumideros es h. El algoritmo propuesto se caracteriza porque se debe encontrar un conjunto de h caminos disyuntos por cada receptor para llevar a cabo la transmisión multicast de máximo flujo. Para lograr este objetivo, se utiliza el algoritmo de etiquetamiento [96] modificado por cada nodo sumidero. En el trabajo, se determina que existen dos tipos de nodos en los caminos de transmisión: nodos de enrutamiento (almacenamiento y reenvío) y nodos claves (nodos de codificación). Sin embargo, los autores no especifican si su propuesta siempre cumple con un esquema de codificación para cualquier h cuando se mezclan los conjuntos de caminos disyuntos por cada sumidero.

Los autores de [97] propusieron una mejora significativa al rendimiento de la transmisión multicast extremo-a-extremo [98], mediante la replicación y reenvío de los datos por nodos superpuestos, en vez de enviarlos por los sistemas extremos. En este trabajo, los nodos superpuestos tienen la capacidad total de codificar y decodificar datos a nivel del mensaje usando LNC; igualmente,

proponen un algoritmo distribuido para construir un Grafo Dirigido Acíclico (*Directed Acyclic Graph* - DAG) correspondiente a la red multicast, en el cual se aplica Network Coding. Este algoritmo duplica el rendimiento extremo-a-extremo en varios casos. En [97], se definen los conceptos de flujo máximo individual y flujo máximo simultáneo, que corresponden, en el modelo planteado de este trabajo de investigación, con el concepto de flujo máximo del grafo unicast y el mínimo flujo máximo del grafo multicast unisesión, respectivamente. Los autores de [97] definen el k-grafo multicast redundante, que corresponde con la definición del grafo multicast unisesión de flujo máximo, donde el flujo máximo k que logra un receptor, es el flujo máximo simultáneo. El algoritmo propuesto parte de un grafo rudimentario, en el cual cada enlace tiene asociado un peso formado por una 2-tupla constituida por el ancho de banda del enlace y su latencia. Luego se pasa a un árbol rudimentario, basado en la técnica del *Spanning Tree* [99]–[102], desde el nodo fuente hasta los nodos intermedios. Del árbol rudimentario, se construye el k-grafo multicast redundante al agregar enlaces del grafo rudimentario, según sea necesario; seleccionando cuidadosamente, al final, los k-caminos, de tal forma que sean disyuntos para cada sumidero.

En [103], se propone un algoritmo para hallar los caminos de transmisión desde un nodo fuente a múltiples receptores en una red de enrutamiento multicast basado en un esquema de codificación en Network Coding. Este algoritmo optimiza la compartición de enlaces en los caminos e incrementa el rendimiento del enrutamiento multicast. También trabaja sobre la base del k-grafo multicast redundante y sobre los k-caminos disyuntos que conforman el flujo máximo entre s y cada nodo sumidero t. La base fundamental de este algoritmo, es la marca de camino (*path stamp*) $ps(n,i,c)$, definido como el hecho que el camino c ha pasado a través del nodo intermedio n para llegar al nodo sumidero $t_i, 1 \leq i \leq |T|$ desde s. Cuando $ps(n,i,c) = 1$, el nodo intermedio n está en el camino desde s hasta t_i, mientras que $ps(n,i,c) = 0$, indica que el nodo no está en el camino. El algoritmo construye el primer camino p_{i1} $(c = 1)$ para cada $t_i, 1 \leq i \leq |T|$ desde s, utilizando el algoritmo del camino más corto Dijkstra modificado. Todos los nodos intermedios del camino son marcados con $ps(n,i,c) = 1$. Se da prioridad a los enlaces no utilizados por otros primeros caminos previos hallados para otro sumidero, con el fin de calcular un nuevo primer camino; es decir, los que tienen $ps(n,j,c) = 0$, donde $j \neq i$. Para caminos posteriores al primero hacia cada t_i, se sigue la misma regla anterior, pero se evitan los enlaces utilizados por los caminos anteriores hacia el mismo t_i; es decir, se evita $ps(n,i,b) = 1$, si $b < c$ para el mismo sumidero t_i. La importancia de este método está en que casi todos los enlaces toman parte en el balanceo de carga del tráfico, y muchos enlaces son compartidos por diferentes caminos desde el nodo fuente hacia los diferentes nodos sumideros.

En [92], [104], se presenta un esquema de enrutamiento y codificación con el objeto de lograr el máximo transporte de paquetes en una sesión multicast. Esto se alcanza mediante la determinación y combinación de los caminos de múltiples flujos máximos hacia cada nodo sumidero para la construcción del grafo de flujo máximo común, utilizando el solapamiento de enlaces comunes en caminos hacia distintos nodos sumideros. Con la determinación del grafo, pretenden alcanzar la tasa de máximo flujo en una red, minimizar el número de nodos de codificación y, en caso de no haber nodos de codificación para una red, el algoritmo dispone de un esquema con los mecanismos tradicionales de reenvío y multicast.

La propuesta presentada en [39], determina que la meta del enrutamiento con Network Coding es construir caminos de enlaces disyuntos desde el nodo fuente hacia cada nodo sumidero. Los autores presentan una solución para el enrutamiento multicast utilizando Network Coding basado en la obtención de las rutas disyuntas por medio del algoritmo "*Packing Steiner Tree*" [24], [25]. Con este método, se obtienen los h-caminos de enlaces disyuntos para cada receptor. Por lo tanto, si se pueden construir h-árboles *Steiner*, se puede garantizar que habrá h-caminos para cada receptor. En [39] se utiliza una modificación del algoritmo Heurística del Camino de Costo Mínimo o *Mínimum Cost Path Heuristic* (MPH) expuesto en [105], [106], el cual utiliza una heurística efectiva para el cálculo del árbol *Steiner*. El algoritmo propuesto evita los enlaces comunes entre los caminos para el mismo receptor mediante una modificación a MPH, de tal forma que se adecúe con Network Coding.

En [107], se propone un esquema distribuido con el fin de llevar a cabo la codificación de red en forma práctica en redes de paquetes reales, alcanzando un rendimiento muy cercano a la capacidad con bajo retardo frente a la pérdida de paquetes, a los cambios de la topología, y la capacidad de los enlaces.

2.5 Conceptos Básicos

2.5.1 Grafo unicast de flujo máximo individual

Sea $G_i' = (V_i', E_i'), i = 1, \dots, |T|$, un grafo unicast que se obtiene a partir del grafo general de comunicaciones G, el cual permite un flujo máximo $f_{G_i'}$ entre s y el nodo sumidero $t_i \in T$. G_i' denota al grafo de flujo máximo individual para el sumidero t_i desde el nodo s.

2.5.2 Mezcla de grafos unicast

Sean $G_i' = (V_i', E_i'), \forall i = 1, \dots, |T|$, $|T|$ grafos dirigidos acíclicos unicast de flujo máximo común f_{max} constituidos con el mismo nodo fuente s y para cada nodo sumideros $t_i \in T$, respectivamente; se define la mezcla de $G_1', G_2', \dots, G_{|T|}'$ como el nuevo grafo dirigido acíclico multicast $G' = (V', E')$ de mínimo flujo máximo común f_{max}, donde

$$V' = \bigcup_{i=1}^{|T|} V_i' \tag{2.5}$$

$$E' = \bigcup_{i=1}^{|T|} E_i' \tag{2.6}$$

El grafo G' calculado contendrá *nodos comunes* y *enlaces comunes*, originados de los distintos caminos constituidos por los G_i' originales que contengan caminos que se solapen en algunos enlaces; es decir, caminos no disyuntos entre grafos distintos. Un nodo u en V' se clasifica como nodo de almacenamiento y reenvío si $\delta_I(u) = 1$, y nodo de codificación cuando $\delta_I(u) > 1$. Si un enlace $e \in E'$ cumple que $\delta_I(o(e)) > 1$, se considera que es un enlace de *cuello de botella*; es decir, el nodo $o(e)$ es un codificador para la red multicast G', y en este nodo debe producirse un nuevo paquete resultante de la codificación de los $\delta_I(o(e))$ paquetes simultáneos entrantes al nodo

$o(e)$ por los enlaces $e' \in \Gamma_I(o(e))$. Si $\delta_I(o(e)) = 1$, el enlace e se considera de reenvío, y transporta la misma información que proviene del único enlace e' entrante al nodo $o(e)$.

2.5.3 Modelo de Enrutamiento Multicast para Network Coding

Tomando como referencia el trabajo de [108], se establece un modelo basado en un grafo dirigido acíclico $G' = (V', E')$, donde V' es el conjunto de nodos en G' y E' es el conjunto de enlaces dirigidos. G' constituye un grafo multicast unisesión de flujo máximo derivado de un grafo de comunicaciones G. En consecuencia, $V' \subseteq V$ y $E' \subseteq E$. En G', el nodo $s \in V'$, se considera el nodo fuente, y $T \subset V'$, se considera el conjunto de nodos sumideros; además $s \notin T$.

Los enlaces en E', se definen como canales de comunicación sin ruido, todos de igual capacidad y de valor igual a una unidad de datos por unidad de tiempo. El nodo fuente tiene una gran cantidad de información que transmitir a los sumideros. La información se divide en paquetes que pertenecen a un alfabeto Σ, y se asume que cada enlace puede transmitir un símbolo de este alfabeto en cada unidad de tiempo. El modelo establece que r es el número máximo de paquetes simultáneos (flujo máximo común de G') que deben ser transmitidos desde el nodo fuente s a cada sumidero t en T. Los paquetes de mensajes están inicialmente presentes en la memoria del nodo fuente.

En [108], se define una función de codificación $\phi_e, e \in E$ según (2.7), para un enlace $e = (u, v)$.

$$\phi_e: \begin{cases} \Sigma^r \longmapsto \Sigma, & u = s \\ \Sigma^{\Gamma_I(u)} \longmapsto \Sigma, & u \neq s \end{cases} \tag{2.7}$$

La función $\phi_{(s,v)}$ determina el paquete transmitido de entre los r-paquetes en el enlace (s, v) o determina el paquete transmitido en el enlace (u, v) para cualquier combinación de paquetes que arriban a los enlaces entrantes en u. En algunas circunstancias, es posible que del nodo fuente, también se obtengan combinaciones de paquetes del alfabeto Σ, como se observará en la sección 5.6.1.

En [108], también se definen los tipos de enlaces para una red con Network Coding, según la función de codificación ϕ_e. Un enlace e se define como un *enlace de codificación*, si ϕ_e depende de dos o más variables (o paquetes entrantes al nodo $o(e)$). Cuando ϕ_e depende de una variable (o paquete entrante al nodo $o(e)$), e es un *enlace de reenvío*.

En el modelo también se define un *enlace de salida* e, cuando emerge directamente del nodo fuente s; es decir, $o(e) = s$, y un *enlace de llegada* e, se caracteriza porque $d(e) = t, t \in T$; es decir, finaliza en un nodo sumidero.

Un esquema de codificación de red para una red multicast G' es factible, si este permite la comunicación con un flujo máximo r entre s y cada sumidero t en T. Es decir, una codificación de red para una red multicast acíclica G', permite la comunicación a una tasa r, si cada sumidero t en T puede calcular los r paquetes originados en el nodo fuente, a partir de los paquetes recibidos en los enlaces entrantes de cada t.

2.5.4 Cálculo de todas las rutas (caminos)

Este algoritmo, que será soporte para la solución de varios problemas de cálculo de rutas, se basa

en la búsqueda en profundidad (DFS) [14], para hallar todos los caminos desde un nodo fuente hasta un nodo sumidero en un grafo acíclico y dirigido G. El algoritmo encuentra todos los caminos posibles existentes entre el nodo fuente y el sumidero, y las devuelve en la estructura vectorial $rutas$, la cual contiene un solo campo denominado $camino$. Se interpreta $rutas(k).camino$, como la existencia del camino número k desde el nodo fuente s hasta el nodo sumidero t_i. Además, el número de caminos desde s hasta t_i corresponde a $|rutas|$.

Ejemplo 2.2: La Figura 2.6 muestra las siete rutas existentes entre el nodo 1 y el nodo 10 en un grafo de comunicaciones de 12 nodos y 3 sumideros. Estas rutas fueron calculadas con el algoritmo de todas las rutas basado en búsqueda en profundidad.

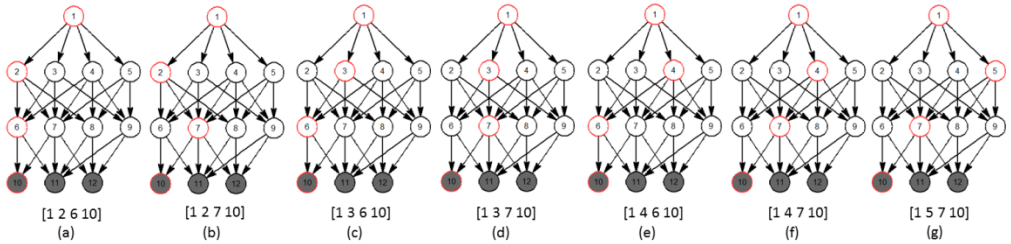


Figura 2.6 Rutas entre el nodo 1 y el nodo 10

2.5.5 Símbolos, códigos y campos

Los enlaces dirigidos de G', pueden transportar de forma confiable paquetes p, donde cada uno es un vector de longitud l bits establecidos sobre el campo finito $\mathbb{F}_q, q = 2^m$, siendo m el número de bits por símbolo en el campo finito. El número L de símbolos de longitud m bits que constituyen el paquete está determinado por:

$$L = \left\lceil \frac{l}{log_2|\mathbb{F}_q|} \right\rceil = \left\lceil \frac{l}{log_2(2^m)} \right\rceil = \left\lceil \frac{l}{m} \right\rceil \tag{2.8}$$

Ejemplo 2.3: Dada una red con un paquete p de 12 bits de longitud y con 3 bits por símbolo (están en el campo $\mathbb{F}_{2^3}$), se pueden deducir los siguientes valores:
La longitud l del paquete es: 12 *bits.*
El número m de bits por símbolo en el campo finito $\mathbb{F}_q$ es: 3 *bits por símbolo.*
El número de símbolos de 3 bits en el campo finito $\mathbb{F}_q$ es: $|\mathbb{F}_q| = 2^3 = 8$ símbolos de 3 bits.
El número L de símbolos en el paquete es: $L = \left\lceil \frac{12\ bits}{3\ bits/símbolo} \right\rceil = 4$ símbolos del campo $\mathbb{F}_q$.
Se observa claramente que es un paquete de longitud 12 bits, $p \in \mathbb{F}_q^4$, con $q = 2^3$. Todos los paquetes que se generen y transiten por los enlaces de la red estarán en el espacio vectorial $\mathbb{F}_{2^3}^4$ y habrá en total $|\mathbb{F}_{2^3}^4|$ paquetes distintos que se pueden generar.
Una muestra de estos paquetes se observa a continuación, donde se agrupan en paréntesis los símbolos de tres bits:

$$p_1 = [(001)(010)(100)(110)]$$
$$p_2 = [(010)(000)(110)(110)]$$
$$p_3 = [(100)(110)(001)(101)]$$

2.5.6 Espacios vectoriales y paquetes

A partir de la tasa de transmisión máxima $r = min_{t \in T} MaxFlujo(s,t)$ entre un nodo fuente s y el conjunto T de nodos sumideros en una transmisión multicast, se establece que r corresponde, en el modelo propuesto, con el número máximo de paquetes que se pueden generar en el nodo fuente s, el cual es el mismo número de paquetes que se entregarán en cada uno de los nodos $t \in T$. Los r paquetes que se generan y entregan son de longitud l símbolos (para el Ejemplo 2.3, los símbolos corresponden a bits; es decir, están en el campo $\mathbb{F}_2$, cuyo tamaño es 2).

Los paquetes se pueden clasificar en *paquetes simples* y *paquetes combinados*. Los primeros corresponden a los r paquetes originales que generaría el computador cliente que está antes del nodo fuente (enrutador fuente s) y que serán obtenidos (a través de decodificación lineal) en los $|T|$ nodos sumideros, mientras que los segundos son resultado de las combinaciones lineales que se producen a lo largo de la transmisión multicast desde el nodo fuente s hasta cualquiera de los nodos sumideros en T.

El nodo fuente s recibe de un computador emisor los r paquetes simples que determinan el máximo flujo de la sesión multicast y, cada paquete simple $p \in \mathbb{F}_2^l$. Por ejemplo, si $l = 7$, los paquetes que se generarán y recibirá s, pertenecen al espacio vectorial $\mathbb{F}_2^7$ y tienen longitud de 7 bits.

Cada paquete, en el modelo, corresponde con una *unidad de información* constituida por l bits, y cada enlace de la red se define con un ancho de banda o capacidad igual a la unidad de información, teniendo todos los enlaces la misma capacidad.

2.6 Planteamiento del Problema

El trabajo desarrollado está encaminado a aportar mecanismos que propenden por la solución de los dos siguientes problemas:

A partir de un grafo G, que representa una red de comunicaciones, conocidos los nodos fuente s y sumideros T de un grupo multicast, el primer problema consiste en hallar el grafo que representa el enrutamiento multicast unisesión G', con el mínimo flujo máximo de caminos disyuntos entre s y cada nodo t en T. El grafo multicast unisesión debe permitir el envío y recepción, en forma simultánea, de hasta el mínimo flujo máximo de paquetes entre el nodo fuente y cada sumidero, sobre la base de utilización de la técnica de Network Coding. A través de esta técnica, se llevan a cabo combinaciones lineales que generan paquetes codificados en los nodos intermedios, de tal forma que puedan ser transmitidos por enlaces cuellos de botella, generando un mayor aprovechamiento del ancho de banda; y además, permite la solución de las combinaciones lineales correspondientes a los paquetes recibidos en los sumideros; solución que producirá los paquetes enviados originalmente por s.

Ejemplo 2.4: La Figura 2.7 (a) muestra un grafo G, que representa una red general de comunicaciones, el cual deriva en un grafo G', mostrado en la Figura 2.7 (b), que representa la red de enrutamiento multicast unisesión de mínimo flujo máximo 2, entre el nodo fuente 1 y los sumideros 10,14,15,16,17 y 18.

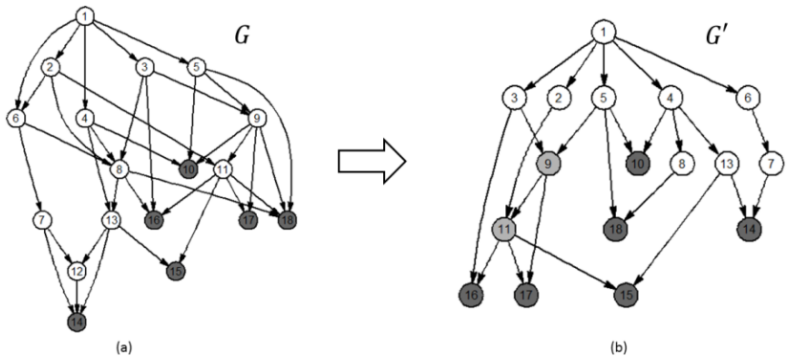


Figura 2.7 (a) Grafo general de comunicaciones. (b) Grafo multicast unisesión resultante sobre la base de NC

Sobre la base del grafo multicast unisesión G', el segundo problema consiste en determinar un esquema de codificación, que especifique si es posible establecer el orden en que el mínimo flujo máximo de paquetes, debe emerger desde el nodo fuente por los enlaces de salida, para que sea posible generar las combinaciones lineales en los nodos intermedios sobre la base de Network Coding, y que conlleve el establecimiento de un sistema lineal de ecuaciones en los nodos sumideros, cuya solución resulte en la entrega simultánea de los paquetes originales enviados por el nodo fuente.

Ejemplo 2.5: La Figura 2.8 representa el grafo G' del ejemplo anterior, con el esquema de codificación establecido sobre la base de Network Coding, donde se muestra el orden de envío de los dos paquetes que constituyen el mínimo flujo máximo, y la recepción simultánea de combinaciones lineales, cuya decodificación conlleva la obtención de los dos paquetes originales.

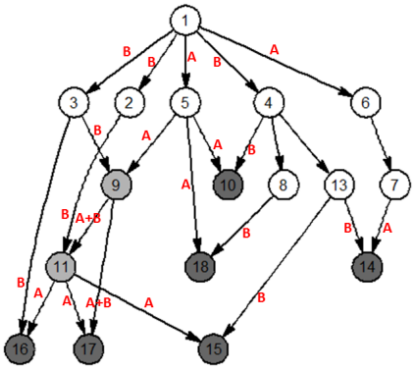


Figura 2.8 Esquema de codificación con un mínimo flujo máximo de 2

2.7 Objetivos

2.7.1 General

Diseñar un protocolo de enrutamiento multicast unisesión que permita la implementación de sistemas lineales de ecuaciones en los nodos intermedios y en los sumideros, sobre la base del paradigma algebraico de Códigos de Red (Network Coding), y cuya solución conlleve la entrega simultánea del mínimo flujo máximo de paquetes originales enviados desde el nodo fuente.

2.7.2 Específicos

- Diseñar, sobre un grafo general de comunicaciones G, una solución aproximada de grafo de enrutamiento multicast unisesión de mínimo flujo máximo G', sobre el cual se pueda aplicar la técnica de codificación/decodificación de Network Coding para el envío y recepción eficiente de paquetes desde el nodo fuente hasta el conjunto de sumideros.
- Definir un esquema de solución de un sistema multicast unisesión, utilizando la técnica de Network Coding, que permita el envío del mínimo flujo máximo de paquetes desde un nodo fuente hasta el conjunto de sumideros.

Capítulo 3

Sesión Multicast para Network Coding: Ford Fullkerson

3.1 Redes de flujo

A partir de los conceptos expuestos en las secciones 2.2, 2.3 y 2.5.3, se construirá el modelo de red de flujo de sesión multicast única que entregará, en igual cantidad, el máximo número de paquetes a los nodos sumideros de la sesión, soportada en el principio de Network Coding.

Ejemplo 3.1: En la Figura 3.1 se observa un grafo con capacidades en los enlaces. Este grafo será utilizado para exponer los ejemplos desarrollados en esta sección. De este grafo se obtienen, para el nodo v_3, los siguientes valores:

$$\Gamma_I(v_3) = \{(v_1, v_3), (v_2, v_3)\}, \delta_I(v_3) = 2$$

$$\Gamma_O(v_3) = \{(v_3, v_4), (v_3, t)\}, \delta_O(v_3) = 2$$

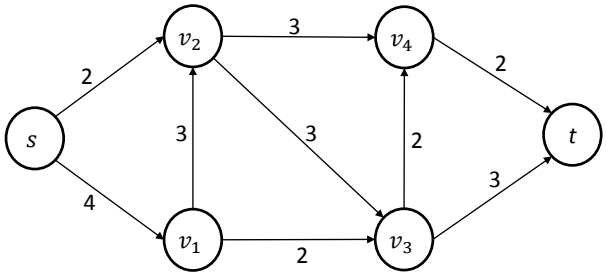


Figura 3.1 Grafo con capacidades

3.1.1 Función Flujo

La función flujo en un grafo de comunicaciones G, se define como:

$$f: V^2 \longmapsto \mathbb{Z} \qquad (3.1)$$
$$f(i,j) = n$$

Esta función satisface las siguientes propiedades [14]:

3.1.1.1 Restricción de capacidad

Significa que el flujo de red en un enlace, no debe exceder la capacidad establecida en el enlace.

$$\forall i,j \in V, 0 \leq f(i,j) \leq C(i,j) \qquad (3.2)$$

3.1.1.2 Simetría inversa

Define que el flujo de red desde un nodo i a un nodo j, es el contrario del flujo de red en la

dirección inversa de j a i.

$$\forall i,j \in V, f(i,j) = -f(j,i) \tag{3.3}$$

Como consecuencia, el flujo de red de un nodo al mismo es 0, ya que $f(i,i) = -f(i,i), \forall i \in V$, luego $f(i,i) = 0$.

3.1.1.3 Conservación del flujo

Este principio determina que el flujo total de red que ingresa a un nodo, es el mismo total que emerge del nodo. Esto aplica dentro del alcance de un nodo particular hasta la red como un todo, lo cual está dentro del marco de las leyes de Kirchhoff [109] para circuitos y nodos.

$$\forall i \in V \setminus \{s,t\}, \sum_{e \in \Gamma_I(i)} f_e = \sum_{e' \in \Gamma_O(i)} f_{e'} \tag{3.4}$$

Sobre el concepto de flujo, también se establecen las siguientes definiciones [14], [110]:

3.1.1.4 Valor del flujo de red entrante a un nodo

El flujo de red positivo entrante en un nodo j se define como:

$$f_j = \sum_{\substack{i \in V \\ f(i,j)>0}} f(i,j) \tag{3.5}$$

Como consecuencia de las dos últimas propiedades, es que el flujo de red positivo que entra a un nodo, distinto de los nodos fuente y sumidero, debe ser igual al flujo de red positivo saliente del nodo.

3.1.1.5 Valor de un flujo

El valor de un flujo f en G se define como:

$$v(f) = \sum_{\substack{s=o(e) \\ u \in V:\, e \in \Gamma_I(u)}} f_e \tag{3.6}$$

lo cual representa el flujo total saliente desde el nodo fuente. Se denomina a G como una red de flujo f.

3.1.1.6 Suma de flujos

Dada una red de flujo $G = (V,E)$, sean f_1 y f_2 funciones de $V^2 \mapsto \mathbb{Z}^+$. La función suma de flujos $f_1 + f_2$ es la función de $V^2 \mapsto \mathbb{Z}^+$ definida por:

$$(f_1 + f_2)(i,j) = f_1(i,j) + f_2(i,j), \forall i,j \in V \tag{3.7}$$

3.1.2 Corte

En general, para un grafo de flujo $G = (V,E)$, un corte [72] es una partición de E en dos subconjuntos U y $\overline{U} = V \setminus U$. También se define un corte como el conjunto de enlaces $e \in E$, tal que $o(e) \in U$ y $d(e) \in \overline{U}$. Un corte s-t se caracteriza porque $s \in U$ y $t \in \overline{U}$.

Ejemplo 3.2: En la Figura 3.2, se observa el corte s-t formado por los enlaces quebrados, donde

$U = \{s, v_1, v_3\}$ y $\overline{U} = \{v_2, v_4, t\}$.

Dado un corte U definido en G con el nodo $i \in U$ y el nodo $j \in \overline{U}$, el enlace (i, j) se denomina enlace hacia adelante, siendo $(U, \overline{U})$ el conjunto de enlaces hacia adelante del corte, y un enlace (i, j) con $i \in \overline{U}$ y $j \in U$, se denomina enlace hacia atrás del corte, siendo $(\overline{U}, U)$ el conjunto de enlaces hacia atrás.

Ejemplo 3.3: De la Figura 3.2, se obtienen los conjuntos:
$(U, \overline{U}) = \{(s, v_2), (v_1, v_2), (v_3, v_4), (v_3, t)\}$ y $(\overline{U}, U) = \{(v_2, v_3)\}$.

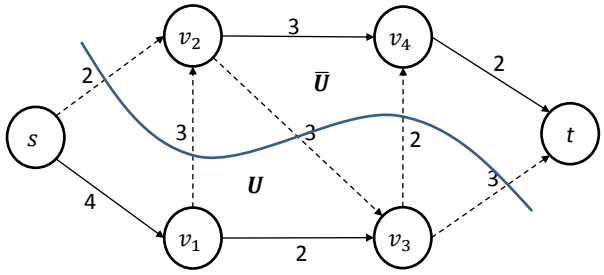


Figura 3.2 Corte s-t

3.1.3 Capacidad del corte s-t

La capacidad $C(U)$ se define como la suma de las capacidades de los enlaces hacia adelante en el corte. Es decir:

$$C(U) = \sum_{(i,j) \in (U, \overline{U})} C(i, j) \tag{3.8}$$

Esta capacidad es la máxima cantidad de flujo que se puede enviar desde los nodos en U hacia los nodos en $\overline{U}$.

Ejemplo 3.4: De la Figura 3.3, $C(U) = C(s, v_2) + C(v_1, v_2) + C(v_3, v_4) + C(v_3, t) = 2 + 3 + 2 + 3 = 10$.

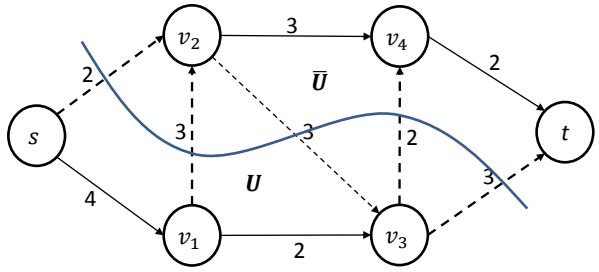


Figura 3.3 Capacidad del corte s-t en G

3.1.4 Mínimo corte

Sea U_T el conjunto de todos los cortes s-t que se pueden obtener en un grafo de flujo $G = (V, E)$, es decir:

$$U_T = \{U | \, U \text{ es un corte } s - t\} \tag{3.9}$$

De igual forma, el conjunto con las capacidades de todos los cortes s-t que se pueden obtener en un grafo de flujo, se representa como:

$$C(U_T) = \{C(U) \mid U \in U_T\} \tag{3.10}$$

El mínimo corte s-t, U_{min} o $MinCorte(s,t)$ se refiere al corte cuya capacidad es la mínima entre todos los cortes s-t. Formalmente, se puede expresar como:

$$U_{min} = MinCorte(s,t) = U : C(U) = minC(U_T) \tag{3.11}$$

3.1.5 Problema del flujo máximo

El problema se plantea en términos de encontrar el flujo $v(f)$ máximo f_G entre el nodo fuente s y el nodo sumidero t en una red representada por el grafo $G = (V, E)$, tal que satisfaga las capacidades de los enlaces y las restricciones de balanceo de flujo en los nodos.

Maximizar:

$$f_G = \sum_{\substack{s=o(e) \\ u \in V : e \in \Gamma_I(u)}} f_e = \sum_{\substack{t=d(e') \\ v \in V : e' \in \Gamma_O(v)}} f_{e'} \tag{3.12}$$

Sujeto a:

$$0 \leq f(i,j) \leq C(i,j), \forall e = (i,j) \in E \tag{3.13}$$

$$\forall i \in V \setminus \{s,t\}, \sum_{e \in \Gamma_I(i)} f_e = \sum_{e' \in \Gamma_O(i)} f_{e'} \tag{3.14}$$

3.1.6 Red residual

Suponiendo que se tiene una red de flujo $G = (V, E)$ y f es el flujo de G. El concepto de red residual está basado en que dado un enlace $e = (i,j) \in E$ con capacidad $C(i,j)$ que transporta un flujo $f(i,j)$, se podría enviar todavía un flujo adicional de $C(i,j) - f(i,j)$ unidades a través del enlace (i,j) antes de exceder la capacidad $C(i,j)$. De igual forma, se pueden enviar hasta $f(i,j)$ unidades de flujo desde el nodo j al nodo i sobre el enlace (i,j), lo cual equivale a cancelar el flujo existente en el enlace.

Dado una red de flujo $G = (V, E)$ y un flujo f, para obtener la red residual, cada enlace $e = (i,j) \in E$ se descompone en dos enlaces $e = (i,j)$ y $e^R = (j,i)$; el enlace $e = (i,j)$ según la Figura 3.4 (a), tiene una capacidad residual

$$C_f(i,j) = C(i,j) - f(i,j) \tag{3.15}$$

y, el enlace $e^R = (j,i)$, según la Figura 3.4 (b), tiene una capacidad residual

$$C_f(j,i) = f(i,j) \tag{3.16}$$

De (3.15) se obtiene (3.16) así:

$$C_f(j,i) = C(j,i) - f(j,i) = 0 - \big(-f(i,j)\big) = f(i,j)$$

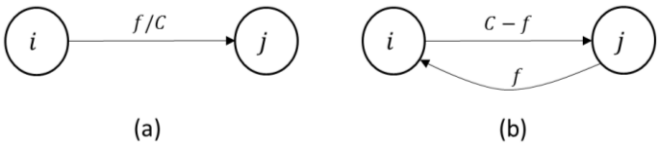


Figura 3.4 (a) Enlace de grafo G. (b) Enlace de grafo G_f.

La red residual de G determinada por f es $G_f = (V, E_f)$, donde

$$E_f = \{(i,j) \in E | \, C_f(i,j) > 0\} \tag{3.17}$$

Ejemplo 3.5: En la Figura 3.5(a) se observa un flujo f en el grafo G de la Figura 3.1 con valor $v(f) = 2$ y en la Figura 3.5(b) se observa la red residual G_f para este flujo.

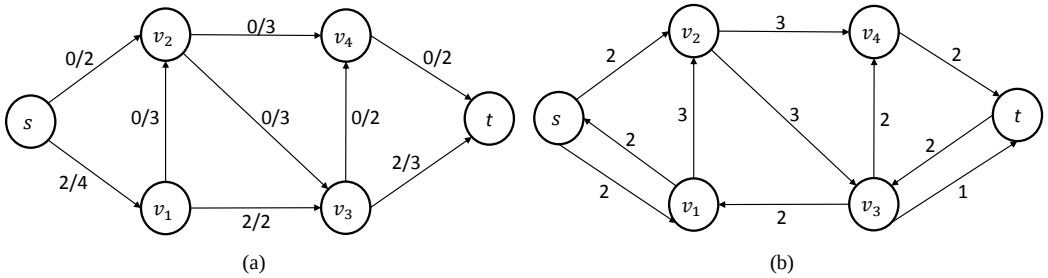


Figura 3.5 (a) Red de flujo G con $v(f) = 2$. (b) Red residual G_f con $v(f) = 2$

El siguiente lema, enunciado y demostrado en [14], determina la relación en los valores de los flujos de G y G_f.

Lema 3.1: Sea $G = (V, E)$ una red de flujo con nodo fuente s y nodo sumidero t, y sea f un flujo en G. Sea G_f la red residual de G inducida por f, y sea f' un flujo en G_f. Luego, la suma de flujos $f + f'$ definida en (3.7) es un flujo en G con valor $v(f + f') = v(f) + v(f')$.

3.1.7 Camino aumentado y enlace saturado

Un camino aumentado π [14], [72] para una red de flujo $G = (V, E)$ y un flujo f, es un camino simple desde el nodo fuente s hasta el nodo sumidero t en una red residual G_f. Acorde con el concepto de red residual, cada enlace (i, j) en π se puede aumentar en un flujo de red positivo adicional desde el nodo i al nodo j sin sobrepasar la restricción de capacidad del enlace. La capacidad residual o capacidad de cuello de botella $b(\pi)$ de un camino aumentado es la mínima capacidad de cualquier enlace en el camino y, se constituye en la máxima cantidad de flujo de red que se puede pasar a lo largo de los enlaces que forman el camino aumentado π. Formalmente, la capacidad residual $b(\pi)$ se define como:

$$b(\pi) = min\{C_f(i,j) | \, (i,j) \text{ está en } \pi\} \tag{3.18}$$

En G_f, un enlace $(i, j) \in E$ se dice saturado, sí y solo sí, $f(i,j) = C(i,j)$ en G ó $C_f(i,j) = 0$ en G_f.

Ejemplo 3.6: En la red residual de la Figura 3.5 (b), el camino aumentado $\pi = s \to v_1 \to v_2 \to v_4 \to t$ tiene una capacidad residual $b(\pi) = min\{C_f(s,v_1), C_f(v_1,v_2), C_f(v_2,v_4), C_f(v_4,t)\} =$

$min\{2,3,3,2\} = 2$. En consecuencia, se pueden pasar 2 unidades de flujo de red adicionales, a través de cada enlace de π, sin sobrepasar la restricción de capacidad.

Lema 3.2: Sea $G = (V, E)$ una red de flujo, sea f un flujo en G, y sea π un camino aumentado en G_f. Se define una función $f_\pi: V^2 \mapsto \mathbb{Z}^+$ por:

$$f_\pi(i,j) = \begin{cases} b(\pi), & \text{si } (i,j) \text{ está en } \pi \\ -b(\pi), & \text{si } (j,i) \text{ está en } \pi \\ 0 & \text{de otro modo} \end{cases} \tag{3.19}$$

Luego, f_π es un flujo en G_f con valor $v(f_\pi) = b(\pi) > 0$.

Demostración:
Para llevar a cabo la demostración que f_π es un flujo en G_f, se deben probar las tres propiedades que debe satisfacer un flujo. Posteriormente, se halla el valor del flujo:

Restricción de capacidad: Por definición de $b(\pi)$, este valor es la mínima capacidad residual en cualquiera de los enlaces a lo largo del camino aumentado π, por tanto la capacidad residual $b(\pi)$ es menor o igual que la capacidad inicial $C(i,j)$ de cualquier enlace (i,j) en el camino aumentado. De lo anterior se desprende que el flujo $f_\pi(i,j) \leq C(i,j)$. Evidentemente, $f_\pi(i,j) = 0 \leq C(i,j)$.

Restricción de simetría inversa: De la definición, se observa que si (i,j) están en π, se satisface que:

Caso I:
$f_\pi(i,j) = b(\pi)$ y, de la definición se puede inferir que $f_\pi(j,i) = -b(\pi)$. Por tanto, $f_\pi(i,j) = -f_\pi(j,i)$.

Caso II:
$f_\pi(i,j) = -b(\pi)$ y, de la definición se puede inferir que $f_\pi(j,i) = b(\pi)$. Por tanto, $f_\pi(i,j) = -f_\pi(j,i)$.

Caso III:
Se cumple que $f_\pi(i,j) = 0 = -f_\pi(j,i)$.

Luego, se cumple siempre que $f_\pi(i,j) = -f_\pi(j,i)$.

Restricción de conservación de flujo: Sea $(i,j),(j,k)$ en π y, dado que $b(\pi)$ es constante a lo largo de π, se deduce que:

$$\sum f_\pi(i,j) = f_\pi(i,j) = b(\pi) = \sum f_\pi(j,k) = f_\pi(j,k) \tag{3.20}$$

Valor del flujo: Dado que el camino aumentado π se define en G_f desde el nodo s, se puede deducir que el enlace (s,j) está en π y $f_\pi(s,j) = b(\pi)$, ya que es un valor constante a lo largo del camino aumentado π, luego:

$$v(f_\pi) = \sum_{j \in V} f_\pi(s,j) = f_\pi(s,j) = b(\pi) > 0 \tag{3.21}$$

Corolario 3.3: Sea $G = (V, E)$ una red de flujo, sea f un flujo en G, y sea π un camino

aumentado en G_f. f_π se define como en (3.19). Además, se define una función $f': V^2 \mapsto \mathbb{Z}^+$ como $f' = f + f_\pi$. Luego, f' es un flujo en G con $v(f') = v(f) + v(f_\pi) > v(f)$.

Demostración:
Del Lema 3.2, se deduce que f_π es un flujo en G_f y del Lema 3.1, se deduce que $f' = f + f_\pi$ es un flujo y que su valor es $v(f') = v(f) + v(f_\pi)$ y, dado que $v(f_\pi) > 0$, también por el Lema 3.2, $v(f') > v(f)$.

3.1.8 Algoritmo de flujo máximo

3.1.8.1 Algoritmo del camino aumentado
A continuación se explica la forma en que el flujo es enviado desde el nodo fuente s hasta el nodo sumidero t en el G_f. Los enlaces en G_f se definen de la siguiente forma:

Definición 3.1: Un enlace (i,j) en un π para un grafo G_f está en el conjunto de enlaces hacia adelante (F), sí y solo sí $f(i,j) < C_f(i,j)$ y, en consecuencia, el flujo f puede ser aumentado.

Definición 3.2: Un enlace (i,j) en un π para un grafo G_f está en el conjunto de enlaces hacia atrás (B), sí y solo sí $f(i,j) > 0$ y, en consecuencia, el flujo f puede ser decrementado.

Sobre la base que se puede enviar un flujo adicional positivo desde s a t cuando se tiene un camino aumentado[14], [72], [110] π en G_f, se construye el Algoritmo 3.1 a continuación:

Algoritmo 3.1: Cálculo del camino **Aumentado**

Entrada: f, π, C_f
Salida: f

```
1  b(π) = min{C_f(i,j) | (i,j) está en π};
2  Para cada enlace (i,j) en π
3     Si (i,j) ∈ F
4        f(i,j) = f(i,j) + b(π);
5     Fin Si
6     Si (i,j) ∈ B
7        f(i,j) = f(i,j) − b(π);
8     Fin Si
9  Fin Para
10 Retornar f;
```

3.1.8.2 Algoritmo de Ford-Fullkerson
Este algoritmo se llama de Ford-Fullkerson [6], [71] en honor a los dos investigadores que lo desarrollaron en 1956 y quiénes probaron el teorema conocido como Máx-Flujo-Min-Corte, el cual determina que el valor del máximo flujo en una red de flujo G es igual a la capacidad del mínimo corte. En 1927, Menger [111], [112] probó una variación para este teorema con grafos no dirigidos con enlaces de capacidad uno, donde siempre existe un conjunto de $r = MinCorte(s,t)$ caminos de enlaces disyuntos entre s y t. En consecuencia, el teorema de Merger y el algoritmo de Ford Fullkerson entregan las bases para obtener de la red una sesión multicast simple [13].

El algoritmo calcula un camino aumentado a través del cual se puede enviar un flujo de red desde el

nodo fuente s hasta el nodo sumidero t en el grafo de red de flujo G. Este camino se usa para enviar tanto flujo como sea posible desde s a t. El proceso se repite hasta que no se pueda encontrar ningún otro camino aumentado, que mejore el flujo total de s a t y, en este caso se ha encontrado el flujo máximo. El algoritmo propuesto determina en un tiempo polinomial el número máximo de caminos con enlaces disyuntos de capacidad unitaria desde s a t. El Algoritmo 3.2 muestra el cálculo del flujo máximo a través de la propuesta de Ford-Fullkerson.

Algoritmo 3.2: Ford-Fullkerson

Entrada: G, s, t, C_f
Salida: f, G_f

```
Para cada enlace (i,j) ∈ E
    f(i,j) = 0;
Fin Para
G_f = G;
Mientras exista un camino aumentado π desde s a t en G_f
    f = Aumentado(f, π, C_f);
    Actualizar G_f;
Fin Mientras
Retornar f, G_f;
```

3.1.9 Problema del mínimo corte

El problema del mínimo corte consiste en encontrar un corte de mínima capacidad entre todos los cortes s-t de un grafo.

3.1.9.1 Capacidad residual de un corte s-t

La capacidad residual $C_f(U)$ de un corte s-t U en G_f es la suma de las capacidades residuales de los enlaces hacia adelante en el corte. A partir de (3.8) se deduce que

$$C_f(U) = \sum_{(i,j)\in(U,\bar{U})} C_f(i,j) \tag{3.22}$$

Ejemplo 3.7: De la Figura 3.6 se puede obtener la capacidad residual en el corte U aplicando (3.22) así $C_f(U) = C_f(s, v_2) + C_f(v_1, v_2) + C_f(v_3, v_4) + C_f(v_3, t) = 2 + 3 + 2 + 1 = 8$.

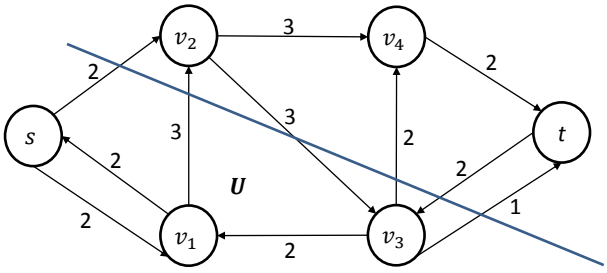


Figura 3.6 Grafo residual G_f con corte U

3.1.9.2 Flujo en un corte s-t

El valor del flujo en una red es el mismo flujo a través de un corte s-t en la red y este se establece a través del siguiente lema:

Lema 3.4: Sea f un flujo en una red G con nodo fuente s, y sea U un corte de G, luego, el flujo de red a través del corte U es $f(U) = v(f)$.

Demostración:

Del principio de conservación del flujo, un nodo $i \in U$ e $i \neq s$, se deduce de (3.4) que

$$\sum_{\substack{e' \in \Gamma_O(i) \\ i \in U}} f_{e'} - \sum_{\substack{e \in \Gamma_I(i) \\ i \in U}} f_e = 0 \tag{3.23}$$

Este principio se puede aplicar a cada nodo $i \in U$, con $i \neq s$ y por tanto se puede establecer la siguiente ecuación a partir de (3.23):

$$\sum_{i \in U} \left(\sum_{e' \in \Gamma_O(i)} f_{e'} - \sum_{e \in \Gamma_I(i)} f_e \right) = 0 \tag{3.24}$$

Se observa que $\forall e' : d(e') \in U, \forall e : o(e) \in U, o(e') = d(e) = i$, los flujos entrantes y salientes del nodo i se anulan. En las sumatorias interiores solo quedaría la diferencia de los flujos de los enlaces que emergen de los nodos en U y de los flujos de los enlaces que ingresan a nodos en U; es decir, la primera sumatoria interior estaría restringida para los enlaces e' salientes de U, tal que $o(e') \in U$ y, la segunda sumatoria interior estaría restringida para los enlaces e entrantes a U, tal que $d(e) \in U$. Esto significa que (3.24) se podría reescribir como:

$$\sum_{e' : o(e') \in U} f_{e'} - \sum_{e : d(e) \in U} f_e = 0 \tag{3.25}$$

Por otro lado, dado que el nodo $s \in U$, al sumar el valor del flujo definido en (3.6) con la anterior ecuación se obtiene que

$$\sum_{\substack{s = o(e) \\ u \in V : e \in \Gamma_I(u)}} f_e + \sum_{e' : o(e') \in U} f_{e'} - \sum_{e : d(e) \in U} f_e = v(f)$$

Los enlaces que nacen en s y que corresponden a la primera sumatoria, se caracterizan porque $o(e) = s$, y por definición de corte s-t, $s \in U$; por tanto, se pueden unir las dos sumatorias iniciales en una sola para simplificar la expresión en

$$v(f) = \sum_{e' : o(e') \in U} f_{e'} - \sum_{e : d(e) \in U} f_e \tag{3.26}$$

La sumatoria derecha significa la cantidad de flujo que sale de los nodos en U hacia los nodos en $\overline{U}$, y la segunda sumatoria es la cantidad de flujo desde los nodos en $\overline{U}$ hacia los nodos en U. Por lo anterior, el flujo a través de cualquier corte s-t es $v(f)$.

A partir de la red residual, se puede obtener el flujo de red $v(f) = f(U)$ a través del corte U, como la suma de los flujos de los enlaces hacia adelante menos la suma de los flujos de los enlaces hacia atrás. Es decir, (3.26) se puede reescribir como:

$$f(U) = v(f) = \sum_{e \in (U,\overline{U})} f_e - \sum_{e \in (\overline{U},U)} f_e \tag{3.27}$$

Ejemplo 3.8: De la Figura 3.7, $f(U) = f(s,v_2) + f(v_1,v_2) + f(v_3,v_4) + f(v_3,t) - f(v_2,v_3) = 0 + 1 + 0 + 3 - 1 = 3$. Este valor corresponde con el $v(f)$ de un paso intermedio antes de obtener el flujo máximo.

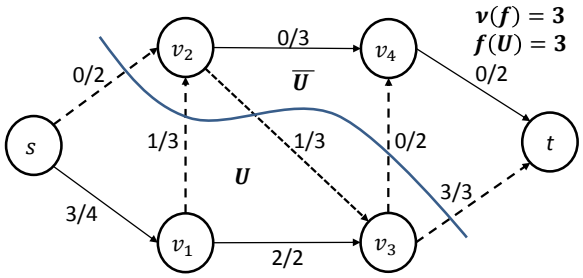


Figura 3.7 Flujos en el corte s-t U de G en un paso intermedio hasta obtener el f_G

Ejemplo 3.9: En la Figura 3.8 se obtiene $f_G = 5$, que corresponde también con $f(U) = f(s,v_1) + f(v_2,v_3) + f(v_4,t) - f(v_1,v_2) - f(v_3,v_4) = 3 + 1 + 2 - 1 - 0 = 5$.

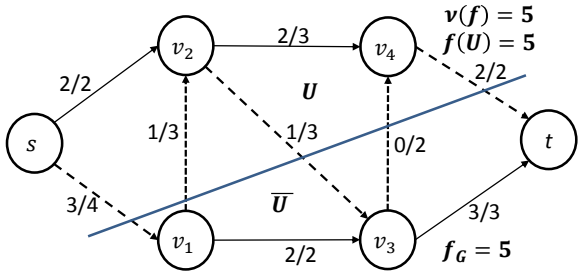


Figura 3.8 Flujos en el corte s-t U de G en el paso final para obtener f_G

3.1.9.3 Capacidad como límite superior del flujo

El siguiente corolario es consecuencia del Lema 3.4, y explica como las capacidades del corte pueden ser usadas para limitar el valor de un flujo.

Corolario 3.5: Sea f cualquier flujo y U un corte s-t cualquiera de G, luego $f_{max} \leq C(U)$.

Demostración:

Tomando a U como cualquier corte s-t de G y sea f cualquier flujo. Por el Lema 3.4 y por (3.2) se tiene que:

$$v(f) = \sum_{e \in (U,\overline{U})} f_e - \sum_{e \in (\overline{U},U)} f_e$$

$$f(U) = v(f) \leq \sum_{e \in (U,\overline{U})} f_e \leq \sum_{e \in (U,\overline{U})} C(e) = C(U)$$

3.1.10 Teorema del Flujo Máximo – Corte Mínimo

Sea f el valor del flujo que es devuelto por el Algoritmo 3.2 para una red de flujo $G = (V, E)$ con nodo fuente s y nodo sumidero t, luego se demostrará que f corresponde con el máximo valor de cualquier flujo en G. También se mostrará que existe un corte s-t U en G para el cual $v(f) = C(U)$ y, como consecuencia, se establece que f tiene el máximo valor sobre cualquier flujo, y que U tiene la mínima capacidad sobre cualquier corte s-t en G. El teorema siguiente muestra estas relaciones:

Teorema 3.6: Si f es un flujo en una red de flujo $G = (V, E)$ con nodo fuente s y nodo sumidero t, entonces se cumplen las siguientes tres propiedades:

1. Existe un corte s-t U en G, tal que la $C(U) = f_{max}$.
2. f_{max} es el flujo máximo de G.
3. No hay caminos aumentados en G_f respecto a f_{max}.

Demostración:

[1⇒2].
Suponemos que U es un corte s-t, tal que $C(U) = f_{max}$ (por 1).
Si existe otro f'_{max}, luego $f'_{max} \leq C(U)$ (por Corolario 3.5).
Entonces, $f'_{max} \leq C(U) = f_{max}$ (por 1).
Entonces, f_{max} es el flujo máximo de G.

[2⇒3] se probará por contradicción [~3⇒~2].
Suponemos que hay un camino aumentado π respecto a f_{max} (por ~3).
Hay un $b(\pi)$ (capacidad residual o de cuello de botella) en el camino aumentado.
Entonces, $f'_{max} = f_{max} + b(\pi)$ (se puede mejorar el f_{max}).
Entonces, f_{max} no es el flujo máximo de G. (Contradicción).

[3⇒1].
Suponemos que no hay caminos aumentados respecto a f_{max} en G_f (por 3).
Se define $U = \{j \in V_f | j \text{ es alcanzable desde } s \text{ por caminos en } G_f\}$.
Se define $\overline{U} = V_f \backslash U$.
Entonces, no existen enlaces en G_f desde U a $\overline{U}$ (Por 3).
Luego en G, todos los enlaces de U a $\overline{U}$ están saturados, es decir $f_e = C(e)$ con $o(e) \in U$ y $d(e) \in \overline{U}$.
Por tanto, se deduce, a partir del Lema 3.4 y por la afirmación anterior, que:

$$v(f_{max}) = \sum_{e \in (U,\overline{U})} f_e - \sum_{e \in (\overline{U},U)} f_e = \sum_{e \in (U,\overline{U})} f_e - 0 = \sum_{e \in (U,\overline{U})} C(e) = C(U)$$

Luego, $f_{max} = C(U)$.

Del anterior teorema, se deriva el siguiente corolario que concluye el teorema del flujo máximo-corte mínimo.

Corolario 3.7: Sea f cualquier flujo, y sea U cualquier corte s-t. Si $v(f) = C(U)$, luego f es máximo y $C(U)$ es mínima.

3.1.10.2 Cálculo del mínimo corte a partir del flujo máximo

Del Corolario 3.7 y, como consecuencia del Teorema 3.6 en los acápites 3) y 1), se puede deducir la forma de obtener el corte mínimo cuando se obtenga el flujo máximo f_G y el grafo de red residual G_f. Sea π_i, el camino desde el nodo fuente s hasta un nodo $i \in V$, donde cada enlace (u, v) en π_i es un enlace hacia adelante, según la Definición 3.1. Formalmente:

$$\pi_i = \{s \rightsquigarrow i \mid (u,v) \text{ está en } s \rightsquigarrow i \iff (u,v) \in F\} \tag{3.28}$$

En el grafo de red residual G_f, con flujo máximo f_G, se obtiene el conjunto

$$U = \{i \in V \mid \exists \pi_i\} \tag{3.29}$$

Es decir, el conjunto U contiene los nodos de G_f alcanzables a partir del nodo fuente s con enlaces hacia adelante. Este conjunto de nodos forma el mínimo corte en G, lo cual quiere decir que la $C(U) = f_{max}$.

Ejemplo 3.10: La Figura 3.9 corresponde con el grafo de red residual G_f de la Figura 3.8 después de aplicar la ecuación (3.28) y, de (3.29), se obtiene el corte mínimo $U = \{s, v_1, v_2, v_3, v_4\}$. La Figura 3.10 muestra el corte mínimo en el grafo de red G donde se observa que la $C(U) = C(v_3, t) + C(v_4, t) = 2 + 3 = 5 = f_G$.

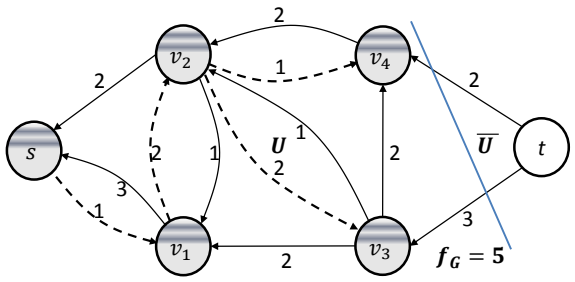


Figura 3.9 Grafo G_f con corte mínimo

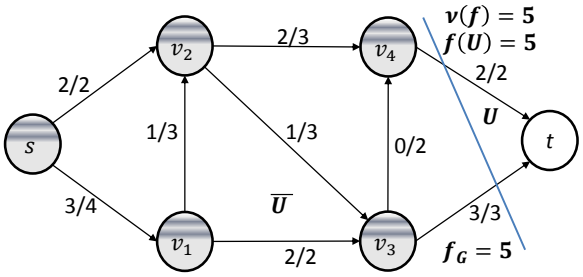


Figura 3.10 Grafo de red G con corte mínimo U

3.2 Solución basada en Ford-Fullkerson

De acuerdo con el comportamiento del Algoritmo de Ford-Fullkerson, se establece un mecanismo para hallar el grafo multicast de mínimo flujo máximo desde un nodo fuente s hasta un grupo de nodos sumideros t_i en T, partiendo de una red de comunicaciones (red de flujo) G. El método planteado permite hallar los $|T|$ grafos de flujo máximo (grafos residuales finales) individuales con

nodo fuente s y nodo sumidero t_i, que luego de un proceso de mezcla de los mismos, alcanza un grafo para comunicación multicast de mínimo flujo máximo G'. El G' obtenido, es una aproximación a un grafo multicast de mínimo flujo máximo, sobre el que se debe demostrar su solubilidad para el envío de paquetes sobre la base de un sistema lineal de ecuaciones soportado en Network Coding. El Algoritmo 3.3 propone el método, el cual consta de los siguientes pasos:

Algoritmo 3.3: Grafo_Sesion_Multicast

Entrada: Arreglos $C, dest$
Salida: Arreglos $flujo$ Escalar $flujomax$

```
1  Sea nn el número de nodos de G o nn = |C|;
2  Para cada i ∈ dest; //i es un nodo sumidero
3     [flumax(i), flujot(i).flujo] = Flujo_Máximo(C, 1, i, nn);
4  Fin Para
5  flumin = min(flumax);
6  Almacenar i en fmpos tal que flumax(i) ≠ flumin;
7  Almacenar flumax(i) en fmval, ∀i ∈ fmpos;
8  Para cada i ∈ fmpos
9     Para j = 1: fmval(i) − flumin
10       flujot = Eliminacion_Camino(flujot, i);
11    Fin Para
12 Fin Para
13 flmin(k, l) = 0, ∀k, l = 1, … , nn;
14 Para cada i ∈ dest
15    Para k = 1, … , nn
16       Para l = 1, … , nn
17          flmin(k, l) = flujot(i).flujo(k, l);
18       Fin Para
19    Fin Para
20 Fin Para
21 nodos_cod = Calculo_Nodos_Codificacion(flmin, dest);
22 Retornar flujo, flujomax;
```

3.2.1 Inicialización del grafo de comunicaciones G

En esta etapa se recibe el grafo G de comunicaciones en la matriz C, donde cada $C(i,j) = 1$, significa que existe un enlace entre el nodo i y el nodo j con capacidad unitaria (un paquete por unidad de tiempo). Cuando $C(i,j) = 0$, no existe un enlace entre i y j.

En la variable $dest$, se almacenan los nodos que actuarán como sumideros, y la variable nn guarda el número total de nodos del grafo G.

3.2.2 Cálculo del flujo máximo para cada sumidero

Entre las líneas 2 a 4 del Algoritmo 3.3, se llama al Algoritmo 3.4 que devuelve el grafo de flujo máximo G_i' para el sumidero $t_i \in T$ dentro de la matriz $flujot(i).flujo$ de tamaño $nn \times nn$, y también devuelve el valor del flujo máximo individual $f_{G_i'}$ desde s hacia t_i, el cual es almacenado en una posición del arreglo de flujos máximos individuales $flumax$. La estructura

vectorial $flujot$ contiene tantas entradas como nodos sumideros existen en T, y cada entrada contiene la matriz de flujo máximo del nodo t_i. Los resultados devueltos, significan que en G_i', representado mediante el arreglo $flujot(i).flujo$ existen máximo $flumax(i)$ caminos disyuntos desde s hasta t_i.

El Algoritmo 3.4, internamente, calcula el grafo de flujo máximo G_i' para el sumidero $t_i \in T$ dentro de la matriz $flujo$ (grafo residual de G para t_i) de tamaño $nn \times nn$. También se calcula el valor del flujo máximo individual $f_{G_i'}$ desde s hacia t_i en la variable $flujomax$.

Los Algoritmos 3.5 y 3.6, cada vez que son llamados, calculan un camino aumentado (π) $camino$, a partir de la matriz de capacidad C (grafo de comunicaciones G), y del estado del $flujo$ en el momento en que es llamado. Este camino es registrado por el Algoritmo 3.4 en la matriz $flujo$ a través de los enlaces que lo constituyen. Como consecuencia del modelo de red que se estableció, donde todos los enlaces de G tienen una capacidad de un paquete por unidad de tiempo, el valor del cuello de botella para cada camino aumentado es 1. Por tanto, para cada enlace hacia adelante, el $flujo$ se incrementa en 1; y para cada enlace hacia atrás, el $flujo$ se decrementa en 1. Además, se incrementa en 1 la variable $flujomax$ cada vez que se registra un camino. Por lo anterior, para cada enlace (u, v) en un camino aumentado, si $C(u, v) = 1$ y $flujo(u, v) = 0$, los valores del $flujo$ se actualizarán como $flujo(u, v) = 1$ y $flujo(v, u) = -1$. Si $C(v, u) = 0$ y $f(v, u) = -1$, los valores del flujo se actualizarán como $flujo(v, u) = 0$ y $flujo(u, v) = 0$. Esto último, determina la anulación del enlace (u, v) en el camino aumentado.

A partir del registro del segundo camino aumentado obtenido, y en adelante, se revisa que no existan nodos comunes entre el último camino calculado y los ya establecidos en la matriz $flujo$. Esto implica un ciclo de revisión en las columnas de la matriz $flujo$, desde la ubicada en la segunda posición hasta la ubicada en la posición previa al sumidero t_i, de tal forma que si hay una columna k con dos celdas $flujo(w_1, k) = 1$ y $flujo(w_2, k) = 1$, se almacenan las posiciones de las filas w_1 y w_2 en el arreglo $posc$.

Si el arreglo $posc$ no está vacío, se elimina el camino aumentado registrado previamente en la matriz $flujo$, reversando los incrementos y decrementos que se llevaron a cabo. También se decrementa en 1 el $flujomax$. Sea (p, k) el enlace del último camino aumentado actualizado tal que p está en $posc$. Se actualiza $C(p, k) = 0$ para evitar que el algoritmo que calcula el camino aumentado (Algoritmo 3.5 o Algoritmo 3.6), tome este enlace nuevamente y, en consecuencia, genere otro camino aumentado distinto al anterior. Es decir, se elimina (p, k) del grafo G original, resultando un nuevo grafo G recortado.

Algoritmo 3.4: Flujo_Máximo

Entrada: Arreglos C Escalar $origen, destino, nn$
Salida: Arreglos $flujo$ Escalar $flujomax$

```
1  flujo(u, v) = 0, ∀u, v = 1, …, nn;
2  flujomax = 0;
3  camino =Aumentado(flujo, C, origen, destino);
4  j = 0;
5  Mientras camino ≠ ∅
```

```
6     Para cada enlace (u, v) ∈ camino
7         flujo(u, v) + +;
8         flujo(v, u) − −;
9     Fin Para
10    flujomax + +;
11    j + +;
12    Si j == 1
13        camino =Aumentado(flujo, C, origen, destino);
14    Fin Si
15    Si j ≥ 2
16        Sea posc el conjunto de posiciones w en flujo^T tal que
              flujo(k, w) == 1 ∧ |flujo^T(k)|_=1 == 2, para algún k = 2, … , destino − 1;
17        Si posc ≠ ∅
18            Para cada enlace (u, v) ∈ camino
19                flujo(u, v) − −;
20                flujo(v, u) + +;
21            Fin Para
22            flujomax − −;
23            p = camino ∩ posc;
24            C(p, k) = 0;
25        Fin Si
26        camino =Aumentado(flujo, C, origen, destino);
27    Fin Si
28 Fin Mientras
29 Retornar flujo, flujomax;
```

3.2.2.1 Cálculo del camino aumentado por Búsqueda en Anchura

El Algoritmo 3.5 calcula un camino aumentado desde el nodo $origen$ (s) hasta un nodo $destino$ (t_i) utilizando el método BFS [14], [113], [114] o búsqueda en anchura (Breadth-First-Search) cada vez que es llamado desde el Algoritmo 3.4. En este algoritmo se observa, en la línea 13, que para el último nodo extraído de la cola q, el nodo c, que forma el enlace (u, c), entra en q cuando no ha sido revisado (está de color BLANCO y $C(u, c) > flujo(u, c)$). Además se cambia su color a GRIS y su predecesor, $pred(c)$ pasa a ser u. La segunda condición en la línea 13 determina que para algún $C(u, c) = 0$ y $flujo(u, c) = -1$ (establecido en el Algoritmo 3.4), se inserte el nodo c en q y se establezca un enlace (u, c) en $camino$ que no respete el orden topológico de los nodos. Es importante destacar, que en este algoritmo, solo se explora desde el nodo $origen$ hasta el nodo $destino$, y no todos los nodos del grafo G representado en la matriz C.

Entre las líneas 20 y 26, se calcula el camino aumentado $camino$, siempre que se cumpla que el color del $destino$ sea NEGRO.

Algoritmo 3.5: Aumentado-BFS

Entrada: Arreglos $flujo, C$ Escalar $origen, destino$
Salida: Arreglos $camino$

```
1  BLANCO = 0;
2  GRIS = 1;
```

```
3  NEGRO = 2;
4  color(i) = BLANCO, ∀i = 1, … , destino;
5  Adicionar origen a la cola q;
6  camino = ∅;
7  color(origen) = GRIS;
8  pred(i) = 0, ∀i = 1, … , destino;
9  Mientras q ≠ ∅
10    Extraer elemento de la cola q y almacenarlo en u;
11    color(u) = NEGRO;
12    Para c = 1: destino
13       Si color(c) == BLANCO ∧ C(u, c) > flujo(u, c)
14          Adicionar c a la cola q;
15          color(c) = GRIS;
16          pred(c) = u;
17       Fin Si
18    Fin Para
19 Fin Mientras
20 Si color(destino) == NEGRO;
21    temp = destino;
22    Mientras pred(temp) ≠ origen;
23       Adicionar pred(temp) a camino;
24       temp = pred(temp);
25    Fin Mientras
26    camino = [origen camino destino];
27 Fin Si
28 Retornar camino;
```

3.2.2.2 Cálculo del camino aumentado por Búsqueda en Profundidad

El Algoritmo 3.6 determina un camino aumentado desde el nodo $origen$ (s) hasta un nodo $destino$ (t_i) utilizando el método DFS[14], [115] o búsqueda en profundidad (Depth-First-Search) cada vez que es llamado por el Algoritmo 3.4. En la línea 6, se llama al Algoritmo 3.7 o Visita-DFS para que construya, a partir del nodo $origen$ (s), la lista de los nodos predecesores y el arreglo de colores de cada nodo hasta el nodo $destino$ (t_i). Luego, se calcula el camino aumentado $camino$ entre las líneas 8 y 11, siempre que se cumpla que el color del $destino$ sea NEGRO.

Algoritmo 3.6: Aumentado-DFS

```
Entrada: Arreglos flujo, C Escalar origen, destino
Salida: Arreglos camino
1  BLANCO = 0;
2  GRIS = 1;
3  NEGRO = 2;
4  color(i) = BLANCO, ∀i = 1, … , destino;
5  pred(i) = 0, ∀i = 1, … , destino;
6  [color, pred] = Visita-DFS(flujo, C, color, pred, origen, destino, BLANCO, GRIS, NEGRO);
7  camino = ∅;
8  Si color(destino) == NEGRO;
```

```
9       temp = destino;
10      Mientras pred(temp) ≠ origen;
11          Adicionar pred(temp) a camino;
12          temp = pred(temp);
13      Fin Mientras
14      camino = [origen camino destino];
15  Fin Si
16  Retornar camino;
```

En cada llamada del Algoritmo 3.7, se cambia el color del nodo raiz u (actúa como $origen$ en cada llamado) a GRIS, y se toman como adyacentes todos los nodos desde el $origen$ hasta el $destino$. Si existe un enlace directo entre el nodo u y el nodo $destino$, se intercambian las posiciones del primer nodo en la lista de adyacencia y la del nodo destino. Esto con el fin de encontrar más rápidamente un camino aumentado hacia el $destino$ y no bloquear otros caminos que puedan conducir posteriormente hacia este nodo, consecuencia del ordenamiento topológico de los nodos.

Entre las líneas 7 a 12, se realiza el ciclo de revisión de los nodos en la lista de adyacencia, si el color del nodo es BLANCO y $C(u,c) > flujo(u,c)$, se asigna como $pred(c)$ a u y se sigue invocando recursivamente la Visita-DFS con el nodo c. Finalmente, después que cada nodo u es explorado, se colorea de NEGRO. Al igual que se especificó en BFS, la segunda condición en la línea 8 determina que para algún $C(u,c) = 0$ y $flujo(u,c) = -1$ (establecido en el Algoritmo 3.4), se establezca un enlace (u,c) en $camino$ que no respete el orden topológico de los nodos. Es importante destacar, que en este algoritmo, solo se explora desde el nodo $origen$ hasta el nodo $destino$, y no todos los nodos del grafo G representado en la matriz C.

Ejemplo 3.11: En la Figura 3.11, si se sigue la búsqueda de los caminos aumentados desde 1 hasta 8 sin realizar el intercambio para los adyacentes al nodo 2, se hallaría como primer camino [1 2 5 8]. Esto conlleva que no se puedan establecer más caminos, ya que cualquier otro no sería disyunto con el primero. En cambio, sí se establece el orden de los nodos adyacentes a 2, colocando en primer lugar al nodo $destino$ 8, se obtendrían los caminos disyuntos en el siguiente orden: [1 2 8] y [1 3 5 8].

Algoritmo 3.7: Visita-DFS

Entrada: Arreglos $flujo, C, color, pred$ Escalar $u, destino$, BLANCO, GRIS, NEGRO
Salida: Arreglos $color, pred$

```
1   color(u) = GRIS;
2   ady(i) = i, ∀i = 1, … , destino;
3   Si C(u, destino) == 1
4       Sea posfin tal que ady(posfin) == destino;
5       Intercambiar ady(1) con ady(posfin);
6   Fin Si
7   Para cada c en ady
8       Si color(c) == BLANCO ∧ C(u,c) > flujo(u,c)
9           pred(c) = u;
10          [color, pred] =Visita-DFS(C, c, color, pred, flujo, destino, BLANCO, GRIS, NEGRO);
```

```
11     Fin Si
12  Fin Para
13  color(u) = NEGRO;
14  Retornar color,pred;
```

Ejemplo 3.12: La Figura 3.11 muestra el grafo de comunicaciones G de 12 nodos, a partir del cual se explicará el funcionamiento del Algoritmo 3.4 para obtener el grafo de flujo máximo G'_{11}. La Figura 3.12 presenta el proceso de construcción utilizando BFS para el cálculo de los caminos aumentados expuesto en el Algoritmo 3.5. Se observa en (a) y (b) que los dos primeros caminos aumentados obtenidos son disyuntos; sin embargo, el camino aumentado obtenido en (c):[1 4 6 9 11], presenta un nodo en común con el camino en (b), por tanto, este último es eliminado. El flujo máximo $flujomax$ toma los valores 1,2 y 3 en (a),(b) y (c), respectivamente. No obstante, por la eliminación del camino en (c), se vuelve a restituir su valor a 2. Además, del grafo de comunicaciones G (matriz de capacidad de la Tabla 3.1(a)), se elimina el enlace (4,6), resultando el grafo G recortado mostrado en (d), correspondiente con la matriz de capacidad de la Tabla 3.1(b). El grafo G recortado, no permitirá al algoritmo de cálculo del camino aumentado volver a generar el camino en (c). En (e), se muestra el cálculo del último camino aumentado sobre el grafo G recortado en (d), resultando el grafo G'_{11}, donde todos los caminos son disyuntos.

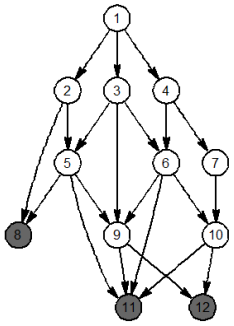


Figura 3.11 Grafo de comunicaciones G de 12 nodos

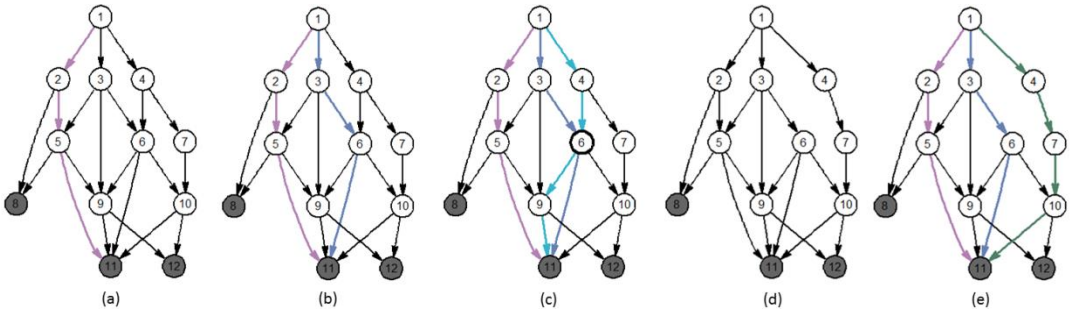


Figura 3.12 Construcción de G'_{11} para G de 12 nodos con Aumentado-BFS

Tabla 3.1 Matrices de capacidades para G de 12 nodos: (a) Original y (b) Recortada

	1	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**
1	0	1	1	1	0	0	0	0	0	0	0	0
2	0	0	0	0	1	0	0	1	0	0	0	0
3	0	0	0	0	1	**1**	0	0	1	0	0	0
4	0	0	0	0	0	**1**	1	0	0	0	0	0
5	0	0	0	0	0	0	0	1	1	0	1	0
6	0	0	0	0	0	0	0	0	1	1	1	0
7	0	0	0	0	0	0	0	0	0	1	0	0

	1	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**
1	0	1	1	1	0	0	0	0	0	0	0	0
2	0	0	0	0	1	0	0	1	0	0	0	0
3	0	0	0	0	1	**1**	0	0	1	0	0	0
4	0	0	0	0	0	**0**	1	0	0	0	0	0
5	0	0	0	0	0	0	0	1	1	0	1	0
6	0	0	0	0	0	0	0	0	0	1	1	0
7	0	0	0	0	0	0	0	0	0	1	0	0

8	0	0	0	0	0	0	0	0	0	0	0
9	0	0	0	0	0	0	0	0	0	1	1
10	0	0	0	0	0	0	0	0	0	1	1
11	0	0	0	0	0	0	0	0	0	0	0
12	0	0	0	0	0	0	0	0	0	0	0

(a)

8	0	0	0	0	0	0	0	0	0	0	0
9	0	0	0	0	0	0	0	0	0	1	1
10	0	0	0	0	0	0	0	0	0	1	1
11	0	0	0	0	0	0	0	0	0	0	0
12	0	0	0	0	0	0	0	0	0	0	0

(b)

Ejemplo 3.13: La Figura 3.13 muestra el proceso de construcción del grafo de flujo máximo G'_{11}utilizando DFS expuesta en el Algoritmo 3.6. En (a), se observa el primer camino hallado. En (b), el segundo camino, tiene el nodo 5 coincidente con el camino en (a). Por tanto, el camino en (b) es eliminado. El flujo máximo $flujomax$ toma los valores 1 y 2 en (a) y (b), respectivamente. Sin embargo, su valor se restituye a 1 después de eliminar el camino. Del grafo G (matriz de capacidad de la Tabla 3.2(a)), se elimina el enlace (3,5), generando el grafo G recortado mostrado en (c), que corresponde con la matriz de capacidad de la Tabla 3.2(b). El grafo recortado se utilizará para establecer el siguiente camino aumentado, el cual ya no tomará el enlace (3,5), como se observa en (d). En (e), se observa que dado que $C(6,3) = 0$ y que por el camino aumentado hallado en (d), el $flujo(6,3) = -1$, se logra calcular el camino aumentado [1 4 6 3 9 11]. Este camino aumentado implicaría la actualización del flujo así: $flujo(6,3) = 0$ y $flujo(3,6) = 0$, lo cual elimina el enlace (3,6) de la matriz de $flujo$. La matriz de $flujo$ resultante se observa en (f), donde el $flujomax = 3$, y se muestra que dos caminos, para alcanzar el nodo sumidero 11, resultan de la combinación de los caminos hallados en (d) y (e).

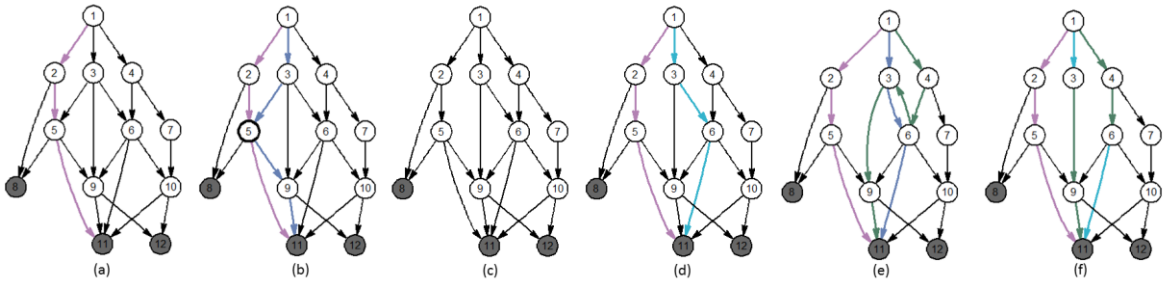


Figura 3.13 Construcción de G'_{11} para G de 12 nodos con Aumentado-DFS

Tabla 3.2 Matrices de capacidades para G de 12 nodos: (a) Original y (b) Recortada

	1	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**
1	0	1	1	1	0	0	0	0	0	0	0	0
2	0	0	0	0	**1**	0	0	1	0	0	0	0
3	0	0	0	0	**1**	1	0	0	1	0	0	0
4	0	0	0	0	0	1	1	0	0	0	0	0
5	0	0	0	0	0	0	0	1	1	0	1	0
6	0	0	0	0	0	0	0	0	1	1	1	0
7	0	0	0	0	0	0	0	0	0	1	0	0
8	0	0	0	0	0	0	0	0	0	0	0	0
9	0	0	0	0	0	0	0	0	0	0	1	1
10	0	0	0	0	0	0	0	0	0	0	1	1
11	0	0	0	0	0	0	0	0	0	0	0	0
12	0	0	0	0	0	0	0	0	0	0	0	0

(a)

	1	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**
1	0	1	1	1	0	0	0	0	0	0	0	0
2	0	0	0	0	**1**	0	0	1	0	0	0	0
3	0	0	0	0	**0**	1	0	0	1	0	0	0
4	0	0	0	0	0	1	1	0	0	0	0	0
5	0	0	0	0	0	0	0	1	1	0	1	0
6	0	0	0	0	0	0	0	0	1	1	1	0
7	0	0	0	0	0	0	0	0	0	1	0	0
8	0	0	0	0	0	0	0	0	0	0	0	0
9	0	0	0	0	0	0	0	0	0	0	1	1
10	0	0	0	0	0	0	0	0	0	0	1	1
11	0	0	0	0	0	0	0	0	0	0	0	0
12	0	0	0	0	0	0	0	0	0	0	0	0

(b)

Ejemplo 3.14: En resumen, en la Figura 3.14, se presentan los grafos de flujo máximo G'_i para cada sumidero en $T = \{8,11,12\}$. Estos grafos se obtuvieron, aplicando BFS para el cálculo de los caminos aumentados sobre el grafo de comunicaciones de la Figura 3.11. Se observa que el grafo en (b) es el mismo de la Figura 3.12 (e) solo con los enlaces utilizados. Además, se observa que

los flujos máximos de estos grafos son, respectivamente (a) 2, (b) 3 y (c) 2. Por tanto, el grafo de (b) se debe llevar a un grafo de flujo máximo 2 para unificar los flujos máximos a uno solo.

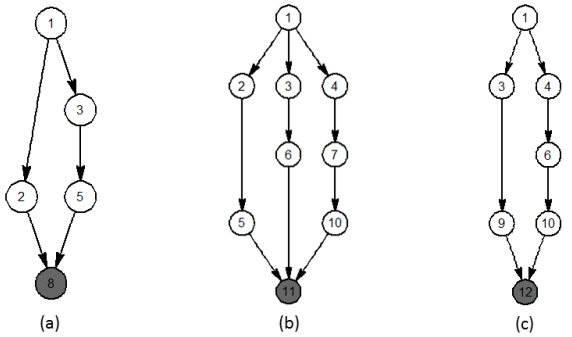


Figura 3.14 Grafos unicast de flujo máximo G_i' para cada $t_i \in T = \{8,11,12\}$ en G de 12 nodos a través de BFS

3.2.3 Igualación de los flujos

En el Algoritmo 3.3, entre las líneas 5 a 12, se ejecuta la igualación de los flujos máximos, de tal forma que todos los grafos G_i' desde el nodo fuente s hasta los sumideros t_i en T, tengan el mismo flujo máximo. Este flujo corresponde, según el teorema del mínimo flujo máximo [5], al mínimo valor ($flumin$) de los flujos almacenados en el arreglo $flumax$.

En el Algoritmo 3.3, los arreglos $fmpos$ y $fmval$ almacenan, respectivamente, los identificadores i y los flujos máximos individuales $flumax(i)$ para cada sumidero t_i que sobrepase el mínimo flujo máximo común $flumin$.

Entre las líneas 7 a 12 del Algoritmo 3.3, se evalúa cada uno de los identificadores de nodos sumideros i almacenados en $fmpos$, hacia los cuales se sobrepasa el mínimo flujo máximo. El objetivo es eliminar los $fmval(i) - flumin$ caminos que conllevan este sobrepaso. Para llevar a cabo esta labor, se llama al procedimiento Eliminación_Camino con los parámetros $flujot$ e i, el cual devuelve el arreglo $flujot$, con la matriz $flujo$ asociada al identificador de nodo sumidero i ajustada al mínimo flujo máximo $flumin$. La Eliminación_Camino se puede ejecutar mediante la eliminación del primer camino del conjunto de caminos (Algoritmo 3.8) o mediante la eliminación del camino más largo (Algoritmo 3.9).

3.2.3.1 Eliminación del primer camino

El Algoritmo 3.8 elimina los primeros caminos, que aparecen en orden topológico, de la lista de caminos que finalizan en el nodo sumidero t_i. Dado que todos los caminos son disyuntos, solo tienen en común el nodo fuente ($s = 1$) y el nodo sumidero t_i. El fin del algoritmo, en cada llamada, es eliminar los enlaces (x, y) que conforman el primero de los caminos que converge en t_i, según el orden topológico de los nodos. Es decir, dados los caminos $c_1: s \to k \to \cdots \to t_i$ y $c_2: s \to l \to \cdots \to t_i$, donde topológicamente, el nodo k está primero que el nodo l o $k < l$, luego el camino c_1 estará primero que el camino c_2.

La variable x, que actúa como el nodo origen de cada enlace, se inicia en el nodo fuente (que para el efecto es el nodo con etiqueta 1), luego se entra al ciclo donde x debe ser diferente del valor de la etiqueta i correspondiente al nodo sumidero. Al interior del ciclo, se busca dentro del

$flujot(i).flujo$ correspondiente al sumidero i en revisión, el nodo destino del enlace saliente de x y se almacena en el arreglo a. Cuando x toma el valor 1, los enlaces salientes serán múltiples como consecuencia de los $flumax(i)$ caminos que se han derivado desde este nodo para alcanzar t_i. Esto conlleva que, en la primera iteración, el arreglo a contenga más de un valor de nodo destino, por tanto, en la línea 4 se asegura tomar el primer nodo destino en la variable y. Para las demás iteraciones, en a solo habrá un valor de nodo destino por la disyunción de los caminos. Lo siguiente es anular el enlace (x, y), y el nodo y pasa de ser destino en el enlace actual a origen en el enlace de la siguiente iteración.

Algoritmo 3.8: Eliminacion_Camino (Primera)

Entrada: Arreglos $flujot$ Escalar i
Salida: Arreglos $flujot$

```
1  x = 1;
2  Mientras x ≠ i
3     Almacenar k en a tal que flujot(i).flujo(x,k) ≠ 0;
4     y = a(1);
5     flujot(i).flujo(x,y) = 0;
6     x = y;
7  Fin Mientras
8  Retornar flujot;
```

3.2.3.2 Eliminación del camino más largo

El Algoritmo 3.9 también se puede utilizar para eliminar caminos de un grafo de flujo máximo G_i'. La característica principal de este algoritmo, es la eliminación del camino más largo del conjunto de $flumax(i)$ caminos que convergen en el sumidero t_i.

El algoritmo recibe el identificador i del sumidero t_i, junto con el arreglo matricial $flujot$. Estas variables, junto con el valor del nodo fuente s ($s = 1$), son utilizados para llamar la función $todas\_las\_rutas$ (sección 2.5.4), la cual devuelve en la estructura vectorial $rutas$, todos los caminos que constituyen el flujo máximo entre s y el sumidero t_i evaluado.

El primer ciclo, entre las líneas 2 y 8, determina en qué posición de $rutas$ está el camino más largo. El segundo ciclo, entre las líneas 9 y 13, elimina el camino más largo, anulando cada uno de sus enlaces (x, y).

Algoritmo 3.9: Eliminacion_Camino (Más Largo)

Entrada: Arreglos $flujot$ Escalar i
Salida: Arreglos $flujot$

```
1  rutas = todas_las_rutas(1, i, flujot(i).flujo);
2  rutmax = 0;
3  Para k = 1:|rutas|
4     Si |rutas(k).camino| > rutmax
5        rutmax = |rutas(k).camino|;
6        posmax = k;
7     Fin Si
8  Fin Para
```

```
9   x = 1;
9   Para l = 2: |rutas(posmax).camino|
10      y = rutas(posmax).camino(l);
11      flujot(i).flujo(x, y) = 0;
12      x = y;
13  Fin Para
14  Retornar flujot;
```

Ejemplo 3.15: El grafo de la Figura 3.14 (b), resultante de BFS, se iguala hasta el mínimo flujo máximo 2, eliminando el primer camino o el camino más largo, según lo previamente explicado. Eliminando el primer camino, se obtiene el grafo de la Figura 3.15(a), y eliminando el camino más largo, se obtiene el grafo de la Figura 3.15(b).

En el primer caso, se elimina el camino [1 2 5 11], ya que en orden topológico, el segundo nodo etiquetado con 2 es menor que 3 y 4, que son los segundos nodos de los caminos restantes. En el segundo caso, se elimina el camino [1 4 7 10 11], ya que es el camino de mayor longitud de los tres que finalizan en 11.

3.2.4 Creación del Grafo de Mínimo Flujo Máximo G'

Entre las líneas 13 a 20 se obtiene el grafo de mínimo flujo máximo G' en la matriz $flmin$ de tamaño $nn \times nn$ a través de la mezcla de los $G'_i, 1 \leq i \leq |T|$. Cada matriz $flujo(i).flujo$ que representa al grafo de flujo máximo G'_i asociado al nodo sumidero t_i, se carga en la matriz $flmin$. Es decir, cada enlace (k, l) existente en un G'_i existirá en G'. La carga de los enlaces que constituyen los caminos de los distintos grafos G'_i, implicará que existan dentro de G' nodos comunes y enlaces comunes, que determinarán, por ende, los nodos de codificación y enlaces cuellos de botella dentro del grafo de red multicast de mínimo flujo máximo.

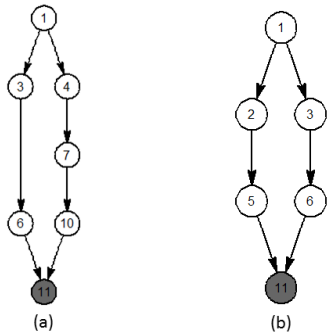


Figura 3.15 Igualación del flujo de G'_{11}, después de aplicar BFS, por Eliminación de (a) Primer Camino, (b) Camino más Largo.

3.2.5 Cálculo de nodos de codificación

El Algoritmo 3.10 calcula los nodos de codificación. El algoritmo revisa si la sumatoria de celdas de cada columna diferente de un nodo sumidero en la matriz $minmaxflujo$ es mayor que 1. Si esto se cumple, el nodo que etiqueta la columna es un codificador. Es importante resaltar, que en los nodos sumideros, también se cumple la condición, por tanto no hay que tenerlos en cuenta, ya que ellos actúan como nodos decodificadores.

Algoritmo 3.10: Cálculo_Nodos_Codificación

Entrada: Arreglos $minmaxflujo, dest$
Salida: Arreglos $nodos\_cod$

```
1  nodos_cod = ∅;
2  Para j = 1:nn
3     Si ∑_{i=1}^{nn} minmaxflujo(i,j) > 1 ∧ j ∉ dest
4        Adicionar j a nodos_cod;
5     Fin Si
6  Fin Para
7  Retornar nodos_cod;
```

Ejemplo 3.16: En los grafos de las Figuras 3.14 (c) y 3.15 (a), se observa que en el nodo 6 finalizan los enlaces (4,6) y (3,6), respectivamente. Esto conlleva tener dos entradas marcadas en 1 en la columna (nodo) 6 de $minmaxflujo$. Por ende, este nodo sería codificador. De igual forma sucede con los enlaces (6,10) y (7,10) de las Figuras 3.14 (c) y 3.15 (a), lo cual conlleva que el nodo 10 también sea codificador. Los enlaces salientes de 6 y 10 se constituyen en cuellos de botella, por ende, dentro de estos nodos, se aplica la codificación de paquetes basado en Network Coding para aprovechar mejor el ancho de banda. La Figura 3.16 (a) muestra el grafo resultante después de aplicar BFS, eliminar el primer camino en el grafo unicast G'_{11}, consolidar los grafos de flujo máximo individuales G'_i en el grafo de mínimo flujo máximo G', y de obtener los nodos de codificación.

Igualmente, en los grafos de la Figuras 3.14 (a) y 3.15 (b), se observa que en el nodo 5 finalizan los enlaces (3,5) y (2,5), respectivamente. También, se observa en las Figuras 3.14 (c) y 3.15 (b), que en el nodo 6 finalizan los enlaces (3,6) y (4,6), respectivamente. Por tanto, los nodos 5 y 6 son codificadores. La Figura 3.16 (b) muestra el grafo resultante después de aplicar BFS, eliminar el camino más largo en el grafo unicast G'_{11}, consolidar los grafos de flujo máximo individuales G'_i en el grafo de mínimo flujo máximo G', y de obtener los nodos de codificación.

Ejemplo 3.17: Siguiendo el mismo proceso de los Ejemplos 3.14 a 3.16, en la Figura 3.17, se obtienen los grafos de flujo máximo G'_i para cada sumidero en $T = \{8,11,12\}$, utilizando DFS para el cálculo de los caminos aumentados sobre el grafo de comunicaciones de la Figura 3.11.

En la Figura 3.18, se muestra la igualación del flujo máximo para el grafo G'_{11}, el cual sobrepasa el mínimo flujo máximo común de 2. La eliminación del primer camino y el camino más largo del grafo en la Figura 3.17 (b), producen el mismo grafo resultante.

En la Figura 3.19, se muestra el grafo resultante después de aplicar DFS, eliminar el camino (primero o más largo) en el grafo unicast G'_{11}, consolidar los grafos de flujo máximo individuales G'_i en el grafo de mínimo flujo máximo G', y de obtener los nodos de codificación.

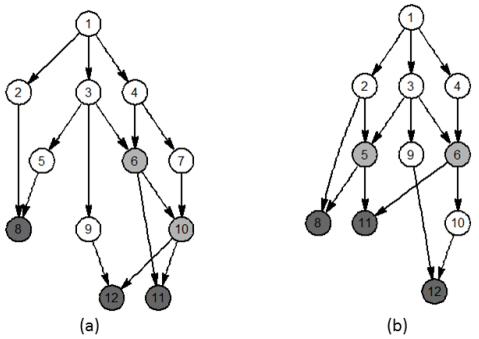


Figura 3.16 Grafo G' de mínimo flujo máximo para G de 12 nodos hallado por BFS y con Eliminación de a) Primer Camino b) Camino más largo

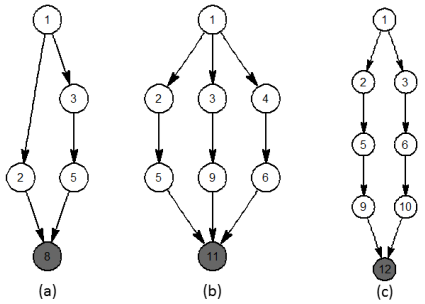


Figura 3.17 Grafos unicast de flujo máximo G_i' para cada $t_i \in T = \{8,11,12\}$ en G de 12 nodos a través de DFS

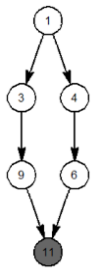


Figura 3.18 Igualación del flujo de G_{11}', después de aplicar DFS, por Eliminación de Primer Camino o Camino más Largo

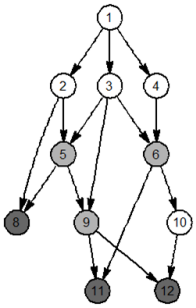


Figura 3.19 Grafo G' de mínimo flujo máximo para G de 12 nodos hallado por DFS

Capítulo 4

Solución de Sesión Multicast para Network Coding: Caminos Disyuntos

En este capítulo se presenta la metodología seguida para la construcción de un grafo de red multicast unisesión de mínimo flujo máximo G' a partir del grafo de comunicaciones G. Por tanto, $G' \preccurlyeq G$, ya que $V' \subseteq V$ y $E' \subseteq E$. El método expuesto se basa en la obtención de caminos disyuntos a partir de los siguientes criterios:

- Para cada nodo sumidero $t_i \in T$, a partir del total de caminos que comunican el nodo fuente s con t_i, obtener el grafo unicast de flujo máximo G_i' entre s y t_i, determinado por el número de caminos disyuntos y de menor longitud que lo constituyen; los cuales corresponden al valor del flujo máximo particular $f_{G_i'}$.
- Hallar el mínimo flujo máximo común $f_{G'}$ para el grafo multicast G' de resolución con Network Coding, aplicando el teorema del mínimo flujo máximo [5].
- Eliminación de los caminos sobrantes en los grafos G_i', que sobrepasen el mínimo flujo máximo común, hasta lograr la igualdad en la cantidad de caminos disyuntos de los $|T|$ grafos G_i' resultantes.
- Construir el grafo multicast G' de mínimo flujo máximo, a partir de la mezcla de los $|T|$ grafos de flujo máximo individuales G_i', que cumplen con el mínimo flujo máximo $f_{G'}$.
- Reducción del grafo de mínimo flujo máximo G' para garantizar o aproximar la resolución del sistema lineal de ecuaciones basado en Network Coding.

El grafo de flujo máximo para una sesión multicast G', según lo descrito en la sección 2.5.3, se construye para llevar a cabo el envío de los paquetes simultáneamente desde el nodo fuente s hasta los nodos sumideros en T utilizando Network Coding. Este grafo se construye a partir del grafo de la red de comunicaciones G.

Según lo anterior, a continuación se presenta el conjunto de procedimientos cuyo objetivo es construir la aproximación de un grafo de flujo máximo G' a partir del grafo de comunicaciones G.

4.1 Grafos de Flujo Máximo de Comunicaciones

En esta sección se describe un procedimiento basado en el cálculo de todos los caminos desde el nodo fuente s hacia cada uno de los nodos sumideros t en T (En algunos apartes, para ser más específico, se denotará como $t_i \in T$). Con los métodos planteados se obtendrán los $|T|$ grafos de flujo máximo individuales con nodo fuente s y nodo sumidero t. Los pasos para lograr la construcción de estos grafos individuales, se explican a continuación.

4.1.1 Inicialización

Esta etapa del algoritmo incluye:

4.1.1.1 Lectura de G

A partir de un archivo de entrada, se lee el grafo G y los nodos sumideros en T. El grafo G corresponde con la matriz de capacidades C, donde cada $C(i,j) = 1$, indica que existe un enlace entre el nodo i y el nodo j con capacidad unitaria. Cuando no exista un enlace entre i y j, se registra un 0.

Ejemplo 4.1: En la gráfica de la Figura 4.1, se muestra un grafo de comunicaciones de 27 nodos, con el nodo fuente etiquetado con 1, y los nodos sumideros 23, 24, 25, 26 y 27. Cada enlace se transforma en una entrada con valor en 1 en la matriz C.

En la Tabla 4.1, se observa la matriz de capacidades C para el ejemplo del grafo G de 27 nodos. Los enlaces entre nodos se destacan con un 1 resaltado en la celda correspondiente.

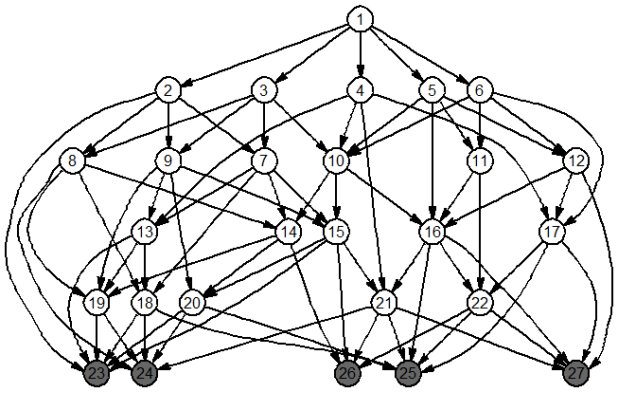


Figura 4.1 Grafo de comunicaciones para 27 nodos y 5 sumideros

Tabla 4.1 Matriz de Capacidades C de G de 27 nodos

C	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27
1	0	**1**	**1**	**1**	**1**	**1**	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
2	0	0	0	0	0	0	**1**	**1**	**1**	0	0	0	0	0	0	0	0	0	0	0	0	0	**1**	0	0	0	0
3	0	0	0	0	0	0	**1**	**1**	**1**	**1**	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
4	0	0	0	0	0	0	0	0	0	**1**	0	0	**1**	0	0	0	**1**	0	0	0	**1**	0	0	0	0	0	0
5	0	0	0	0	0	0	0	0	0	**1**	**1**	**1**	0	0	0	**1**	0	0	0	0	0	0	0	0	0	0	0
6	0	0	0	0	0	0	0	0	0	**1**	**1**	**1**	0	0	0	0	**1**	0	0	0	0	0	0	0	0	0	0
7	0	0	0	0	0	0	0	0	0	0	0	0	**1**	**1**	**1**	0	0	**1**	0	0	0	0	0	0	0	0	0
8	0	0	0	0	0	0	0	0	0	0	0	0	0	**1**	0	0	0	**1**	**1**	0	0	0	0	**1**	0	0	0
9	0	0	0	0	0	0	0	0	0	0	0	0	**1**	0	**1**	0	0	0	**1**	**1**	0	0	0	0	0	0	0
10	0	0	0	0	0	0	0	0	0	0	0	0	0	**1**	**1**	**1**	0	0	0	0	0	0	0	0	0	0	0
11	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	**1**	0	0	0	0	0	**1**	0	0	0	0	0
12	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	**1**	**1**	0	0	0	0	0	0	0	0	0	**1**
13	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	**1**	**1**	0	0	0	**1**	0	0	0	0
14	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	**1**	**1**	0	0	0	0	0	0	0
15	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	**1**	**1**	0	**1**	0	0	**1**	0
16	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	**1**	**1**	0	0	**1**	0	**1**
17	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	**1**	0	0	**1**	0	**1**
18	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	**1**	**1**	**1**	0	0
19	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	**1**	**1**	0	0	0
20	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	**1**	**1**	**1**	0	0
21	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	**1**	**1**	**1**	**1**
22	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	**1**	**1**	**1**
23	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
24	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
25	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
26	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
27	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

4.1.1.2 Cálculo de valores iniciales

Se calcula el número de nodos de G (nn) y el número de sumideros $(ndest)$. La lista de segundos nodos o nodos adyacentes al nodo fuente, se guardan en un arreglo unidimensional $(segnodtot)$. Se lleva un conteo del uso de los segundos nodos por cada camino desde el nodo fuente hasta cada nodo sumidero en un arreglo unidimensional. Se inicializa el arreglo que contendrá el número de caminos válidos posibles o flujo máximo $(flumaxrut)$ por cada sesión entre el nodo fuente s y cada sumidero $t_i \in T$; es decir, por cada sumidero t_i habrá una entrada $flumaxrut(i)$, por tanto el tamaño de este arreglo es $ndest$. El índice i es la posición que ocupa el nodo sumidero en el arreglo $dest$.

Ejemplo 4.2: Según la Figura 4.1, estos valores corresponden con los presentados en la Tabla 4.2.

Tabla 4.2 Variables y valores iniciales para G de 27 nodos

Variable	**Valores Iniciales**
Número de nodos de G (nn)	27
Nodo fuente (s)	1
Nodos sumideros $(dest)$	[23, 24, 25, 26, 27]
Número de sumideros de G $(ndest)$	5
Segundos nodos adyacentes a s $(segnodtot)$	[2, 3, 4, 5, 6]
Contador de uso de segundos nodos $(contsegnod)$	[0, 0, 0, 0, 0]
Flujo máximo o Caminos posibles entre s y cada $t_i \in T(flumaxrut)$	[]

4.1.2 Cálculo de grafos individuales de flujo máximo

Se denota por nd el índice con el que se recorre el arreglo $dest$ y, por ende, $1 \leq nd \leq ndest$. Sea el nodo sumidero $t \in T$, luego existe un nd tal que $dest(nd) = t$. Para cada $t \in T$, se llevan a cabo los siguientes cálculos:

4.1.2.1 Cálculo de todos los caminos

Se aplica un algoritmo de cálculo de todos los caminos (sección 2.5.4) desde el nodo fuente s hasta el nodo t. Estos caminos se guardan en una estructura unidimensional $(rutas)$, donde el índice de este vector corresponde con el identificador de camino. En total, existe un número de caminos almacenado en $nrutas$, que es igual a la longitud del vector $rutas$. Cada camino i se especifica como $rutas(i)$, y es desplegado como la secuencia de nodos que lo representan, desde el nodo fuente s hasta el nodo sumidero t, en el arreglo $rutas(i).camino$ con la siguiente forma:

$$rutas(i).camino = [s, a_1, a_2, \dots, a_\ell, t] \tag{4.1}$$

Donde los $a_k, k = 1, \dots, \ell$ corresponden con los ℓ nodos que deben ser previamente visitados para llegar al nodo t desde el nodo s.

Ejemplo 4.3: A partir de la Tabla 4.3 hasta la Tabla 4.7, se muestra la estructura de los caminos para los sumideros del grafo de comunicaciones de la Figura 4.1. En estas tablas se observan los caminos existentes desde el nodo fuente 1 hasta cada sumidero, indexadas por el contador i en orden secuencial. De estos caminos, quedarán seleccionados los que sean más cortos y que tengan

un menor número de colisiones, como se explicará más adelante. De igual forma, se pueden obtener los caminos hacia cada uno de los restantes sumideros. En la Tabla 4.8, se puede observar el número de caminos posibles entre el nodo fuente 1 y cada nodo sumidero en este ejemplo.

Tabla 4.3 Caminos desde el nodo fuente 1 al sumidero 23 en G de 27 nodos

i	$rutas(i).camino$	i	$rutas(i).camino$	i	$rutas(i).camino$	i	$rutas(i).camino$
1	1 2 7 13 18 23	15	1 2 9 13 23	29	1 3 8 14 19 23	43	1 3 10 15 23
2	1 2 7 13 19 23	16	1 2 9 15 20 23	30	1 3 8 14 20 23	44	1 4 10 14 19 23
3	1 2 7 13 23	17	1 2 9 15 23	31	1 3 8 18 23	45	1 4 10 14 20 23
4	1 2 7 14 19 23	18	1 2 9 19 23	32	1 3 8 19 23	46	1 4 10 15 20 23
5	1 2 7 14 20 23	19	1 2 9 20 23	33	1 3 9 13 18 23	47	1 4 10 15 23
6	1 2 7 15 20 23	20	1 2 23	34	1 3 9 13 19 23	48	1 4 13 18 23
7	1 2 7 15 23	21	1 3 7 13 18 23	35	1 3 9 13 23	49	1 4 13 19 23
8	1 2 7 18 23	22	1 3 7 13 19 23	36	1 3 9 15 20 23	50	1 4 13 23
9	1 2 8 14 19 23	23	1 3 7 13 23	37	1 3 9 15 23	51	1 5 10 14 19 23
10	1 2 8 14 20 23	24	1 3 7 14 19 23	38	1 3 9 19 23	52	1 5 10 14 20 23
11	1 2 8 18 23	25	1 3 7 14 20 23	39	1 3 9 20 23	53	1 5 10 15 20 23
12	1 2 8 19 23	26	1 3 7 15 20 23	40	1 3 10 14 19 23	54	1 5 10 15 23
13	1 2 9 13 18 23	27	1 3 7 15 23	41	1 3 10 14 20 23	55	1 6 10 14 19 23
14	1 2 9 13 19 23	28	1 3 7 18 23	42	1 3 10 15 20 23	56	1 6 10 14 20 23
						57	1 6 10 15 20 23
						58	1 6 10 15 23

Tabla 4.4 Caminos desde el nodo fuente 1 al sumidero 24 en G de 27 nodos

i	$rutas(i).camino$	i	$rutas(i).camino$	i	$rutas(i).camino$	i	$rutas(i).camino$
1	1 2 7 13 18 24	17	1 2 9 19 24	33	1 3 9 15 20 24	49	1 4 21 24
2	1 2 7 13 19 24	18	1 2 9 20 24	34	1 3 9 15 21 24	50	1 5 10 14 19 24
3	1 2 7 14 19 24	19	1 3 7 13 18 24	35	1 3 9 19 24	51	1 5 10 14 20 24
4	1 2 7 14 20 24	20	1 3 7 13 19 24	36	1 3 9 20 24	52	1 5 10 15 20 24
5	1 2 7 15 20 24	21	1 3 7 14 19 24	37	1 3 10 14 19 24	53	1 5 10 15 21 24
6	1 2 7 15 21 24	22	1 3 7 14 20 24	38	1 3 10 14 20 24	54	1 5 10 16 21 24
7	1 2 7 18 24	23	1 3 7 15 20 24	39	1 3 10 15 20 24	55	1 5 11 16 21 24
8	1 2 8 14 19 24	24	1 3 7 15 21 24	40	1 3 10 15 21 24	56	1 5 12 16 21 24
9	1 2 8 14 20 24	25	1 3 7 18 24	41	1 3 10 16 21 24	57	1 5 16 21 24
10	1 2 8 18 24	26	1 3 8 14 19 24	42	1 4 10 14 19 24	58	1 6 10 14 19 24
11	1 2 8 19 24	27	1 3 8 14 20 24	43	1 4 10 14 20 24	59	1 6 10 14 20 24
12	1 2 8 24	28	1 3 8 18 24	44	1 4 10 15 20 24	60	1 6 10 15 20 24
13	1 2 9 13 18 24	29	1 3 8 19 24	45	1 4 10 15 21 24	61	1 6 10 15 21 24
14	1 2 9 13 19 24	30	1 3 8 24	46	1 4 10 16 21 24	62	1 6 10 16 21 24
15	1 2 9 15 20 24	31	1 3 9 13 18 24	47	1 4 13 18 24	63	1 6 11 16 21 24
16	1 2 9 15 21 24	32	1 3 9 13 19 24	48	1 4 13 19 24	64	1 6 12 16 21 24

Tabla 4.5 Caminos desde el nodo fuente 1 al sumidero 25 en G de 27 nodos

i	$rutas(i).camino$	i	$rutas(i).camino$	i	$rutas(i).camino$	i	$rutas(i).camino$
1	1 2 7 13 18 25	19	1 3 9 13 18 25	37	1 4 17 25	55	1 5 16 22 25
2	1 2 7 14 20 25	20	1 3 9 15 20 25	38	1 4 21 25	56	1 5 16 25
3	1 2 7 15 20 25	21	1 3 9 15 21 25	39	1 5 10 14 20 25	57	1 6 10 14 20 25
4	1 2 7 15 21 25	22	1 3 9 20 25	40	1 5 10 15 20 25	58	1 6 10 15 20 25
5	1 2 7 18 25	23	1 3 10 14 20 25	41	1 5 10 15 21 25	59	1 6 10 15 21 25
6	1 2 8 14 20 25	24	1 3 10 15 20 25	42	1 5 10 16 21 25	60	1 6 10 16 21 25
7	1 2 8 18 25	25	1 3 10 15 21 25	43	1 5 10 16 22 25	61	1 6 10 16 22 25
8	1 2 9 13 18 25	26	1 3 10 16 21 25	44	1 5 10 16 25	62	1 6 10 16 25
9	1 2 9 15 20 25	27	1 3 10 16 22 25	45	1 5 11 16 21 25	63	1 6 11 16 21 25
10	1 2 9 15 21 25	28	1 3 10 16 25	46	1 5 11 16 22 25	64	1 6 11 16 22 25
11	1 2 9 20 25	29	1 4 10 14 20 25	47	1 5 11 16 25	65	1 6 11 16 25
12	1 3 7 13 18 25	30	1 4 10 15 20 25	48	1 5 11 22 25	66	1 6 11 22 25
13	1 3 7 14 20 25	31	1 4 10 15 21 25	49	1 5 12 16 21 25	67	1 6 12 16 21 25
14	1 3 7 15 20 25	32	1 4 10 16 21 25	50	1 5 12 16 22 25	68	1 6 12 16 22 25
15	1 3 7 15 21 25	33	1 4 10 16 22 25	51	1 5 12 16 25	69	1 6 12 16 25
16	1 3 7 18 25	34	1 4 10 16 25	52	1 5 12 17 22 25	70	1 6 12 17 22 25
17	1 3 8 14 20 25	35	1 4 13 18 25	53	1 5 12 17 25	71	1 6 12 17 25
18	1 3 8 18 25	36	1 4 17 22 25	54	1 5 16 21 25	72	1 6 17 22 25
						73	1 6 17 25

Tabla 4.6 Caminos desde el nodo fuente 1 al sumidero 26 en G de 27 nodos

i	$rutas(i).camino$	i	$rutas(i).camino$	i	$rutas(i).camino$	i	$rutas(i).camino$
1	1 2 7 14 26	13	1 3 10 14 26	25	1 5 10 14 26	37	1 5 16 22 26
2	1 2 7 15 21 26	14	1 3 10 15 21 26	26	1 5 10 15 21 26	38	1 6 10 14 26
3	1 2 7 15 26	15	1 3 10 15 26	27	1 5 10 15 26	39	1 6 10 15 21 26
4	1 2 8 14 26	16	1 3 10 16 21 26	28	1 5 10 16 21 26	40	1 6 10 15 26
5	1 2 9 15 21 26	17	1 3 10 16 22 26	29	1 5 10 16 22 26	41	1 6 10 16 21 26
6	1 2 9 15 26	18	1 4 10 14 26	30	1 5 11 16 21 26	42	1 6 10 16 22 26
7	1 3 7 14 26	19	1 4 10 15 21 26	31	1 5 11 16 22 26	43	1 6 11 16 21 26
8	1 3 7 15 21 26	20	1 4 10 15 26	32	1 5 11 22 26	44	1 6 11 16 22 26
9	1 3 7 15 26	21	1 4 10 16 21 26	33	1 5 12 16 21 26	45	1 6 11 22 26
10	1 3 8 14 26	22	1 4 10 16 22 26	34	1 5 12 16 22 26	46	1 6 12 16 21 26
11	1 3 9 15 21 26	23	1 4 17 22 26	35	1 5 12 17 22 26	47	1 6 12 16 22 26
12	1 3 9 15 26	24	1 4 21 26	36	1 5 16 21 26	48	1 6 12 17 22 26
						49	1 6 17 22 26

Tabla 4.7 Caminos desde el nodo fuente 1 al sumidero 27 en G de 27 nodos

i	$rutas(i).camino$	i	$rutas(i).camino$	i	$rutas(i).camino$	i	$rutas(i).camino$
1	1 2 7 15 21 27	13	1 4 17 22 27	25	1 5 12 16 22 27	37	1 6 11 16 21 27
2	1 2 9 15 21 27	14	1 4 17 27	26	1 5 12 16 27	38	1 6 11 16 22 27
3	1 3 7 15 21 27	15	1 4 21 27	27	1 5 12 17 22 27	39	1 6 11 16 27
4	1 3 9 15 21 27	16	1 5 10 15 21 27	28	1 5 12 17 27	40	1 6 11 22 27
5	1 3 10 15 21 27	17	1 5 10 16 21 27	29	1 5 12 27	41	1 6 12 16 21 27
6	1 3 10 16 21 27	18	1 5 10 16 22 27	30	1 5 16 21 27	42	1 6 12 16 22 27
7	1 3 10 16 22 27	19	1 5 10 16 27	31	1 5 16 22 27	43	1 6 12 16 27
8	1 3 10 16 27	20	1 5 11 16 21 27	32	1 5 16 27	44	1 6 12 17 22 27
9	1 4 10 15 21 27	21	1 5 11 16 22 27	33	1 6 10 15 21 27	45	1 6 12 17 27
10	1 4 10 16 21 27	22	1 5 11 16 27	34	1 6 10 16 21 27	46	1 6 12 27
11	1 4 10 16 22 27	23	1 5 11 22 27	35	1 6 10 16 22 27	47	1 6 17 22 27
12	1 4 10 16 27	24	1 5 12 16 21 27	36	1 6 10 16 27	48	1 6 17 27

Tabla 4.8 Número de caminos por cada sumidero en G de 27 nodos

Nodo sumidero t	**Número de caminos $nrutas$**
23	58
24	64
25	73
26	49
27	48

4.1.2.2 Cálculo de la matriz $posi$

La matriz $posi$ de tamaño $nrutas \times nn$, se calcula para cada sumidero t. Cada posición de esta matriz indica si un nodo j $(1 \leq j \leq nn)$ se encuentra en un camino i $(1 \leq i \leq nrutas)$ que conduce desde s a t. Esta matriz será relevante al momento del cálculo de las colisiones. En consecuencia, cada posición de la matriz $posi$ asociada al sumidero $t \in T$, almacena la información siguiente:

$$posi(i,j) = \begin{cases} 1, & \text{si el nodo } j \text{ está en el camino } i \text{ desde } s \text{ a } t \\ 0, & \text{si el nodo } j \text{ no está en el camino } i \text{ desde } s \text{ a } t \end{cases} \quad (4.2)$$

Inicialmente la matriz $posi$ se inicializa con ceros en todas sus posiciones. Para el caso del nodo sumidero 23, en el ejemplo en revisión, la matriz $posi$ tiene una dimensión de 58×27.

4.1.2.3 Cálculo del vector de longitudes de caminos $long$

Este vector contiene la longitud de cada camino encontrado desde s hasta el sumidero t. El tamaño del vector $long$ es $nrutas$.

Ejemplo 4.4: Para el grafo G con 27 nodos, en la Tabla 4.9 se muestran los vectores de longitudes para cada lista de caminos de cada nodo sumidero.

Tabla 4.9 Longitudes de los caminos que conducen a sumideros en G de 27 nodos

t	$long$
23	[6,6,5,6,6,6,5,5,6,6,5,5,6,6,5,6,5,5,5,3,6,6,5,6,6,6,5,5,6,6,5,5,6,6,5,6,5,5,5,5,6,6,6,5,6,6,6,5,5,5,4,6,6,6,5,6,6,6,5]
24	[6,6,6,6,6,6,5,6,6,5,5,4,6,6,6,6,6,5,5,6,6,6,6,6,6,5,6,6,5,5,4,6,6,6,6,6,5,5,6,6,6,6,6,6,6,6,6,5,5,4,6,6,6,6,6,6,6,5,6,6,6,6,6,6,6]
25	[6,6,6,6,6,5,6,5,6,6,6,6,5,6,6,6,6,6,5,6,5,6,6,6,6,5,6,6,5,6,6,6,6,6,6,5,6,6,6,6,6,6,5,5,5,5,4,4,6,6,6,6,6,5,6,6,5,5,6,6,5,6,5,5,5,4,6,6,6,6,6,5,6,6,5,5,6,6,5,6,5,5,4]
26	[5,6,5,5,6,5,5,5,6,5,5,6,5,5,6,5,6,6,5,6,5,6,6,5,4,5,6,5,6,6,6,6,5,6,6,6,5,5,5,6,5,6,6,6,6,5,6,6,6,5]
27	[6,6,6,6,6,6,6,5,6,6,6,5,5,4,4,6,6,6,5,6,6,5,5,6,6,5,6,5,4,5,5,4,6,6,6,5,6,6,5,5,6,6,5,6,5,5,4,5,4]

4.1.2.4 Cálculo de los nodos visitados por todos los caminos desde s hasta t

Los identificadores de los caminos que pasan por cada nodo desde el nodo fuente hasta el sumidero t, son guardados en la estructura vectorial $nodosenruta$, la cual tiene la forma $nodosenruta(j).lista$. Esto significa que para el nodo j (excepto el nodo s y el sumidero t) se guardan en el vector $lista$ los identificadores de los caminos hallados y registrados en el vector $rutas$ para el sumidero t en revisión. Si el i-ésimo $camino$ de $rutas$ pasa por el nodo j, luego $nodosenruta(j).lista$ contiene al identificador i.

El Algoritmo 4.1 muestra la forma en que se registran los identificadores de caminos j almacenados en el vector $rutas$ dentro de la estructura vectorial $nodosenruta(i).lista$, donde $2 \leq i \leq t-1$. La estructura vectorial $nodosenruta$, se construye a través de dos iteraciones: la primera, sobre las columnas de $posi$ iniciando en el segundo nodo de esta matriz y terminando en una posición previa al sumidero y, la segunda, sobre todos los caminos que conforman las filas de $posi$. El tamaño de $nodosenruta$ es $t-2$.

Ejemplo 4.5: Para G de 27 nodos, desde la Tabla 4.10 hasta la Tabla 4.14, se presentan los resultados del cálculo de los nodos visitados por todos las caminos desde el nodo fuente 1 hasta los nodos sumideros 23, 24, 25, 26 y 27. Se observa en la Tabla 4.10, que el nodo 3 es el más visitado por los caminos que se dirigen desde 1 hasta 23. También se observa que los nodos 11, 12, 16, 17, 21 y 22 nunca son visitados por ningún camino y su $lista$ está vacía. El mismo análisis se puede realizar para los demás nodos sumideros y sus respectivas tablas.

Algoritmo 4.1: Cálculo_Caminos_Atravesando_Cada_Nodo

Entrada: Arreglos $posi, dest, nd$, Escalar $nrutas$
Salida: Estructura vectorial $nodosenruta$

```
1  Para j = 2: dest(nd) − 1
2      k = 1;
3      Para i = 1: nrutas
4          Si posi(i, j) == 1
5              nodosenruta(j).lista(k) = i;
6              k++;
7          Fin Si
8      Fin Para
9  Fin Para
10 Retornar nodosenruta;
```

Tabla 4.10 Listado de nodos visitados por los caminos desde 1 hasta 23 en G de 27 nodos

Nodos visitados por todos los caminos desde el fuente hasta el sumidero

Caminos	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22
1	1	21	44	51	55	1	9	13	40			1	4	6			1	2	5		
2	2	22	45	52	56	2	10	14	41			2	5	7			8	4	6		
3	3	23	46	53	57	3	11	15	42			3	9	16			11	9	10		
4	4	24	47	54	58	4	12	16	43			13	10	17			13	12	16		
5	5	25	48			5	29	17	44			14	24	26			21	14	19		
6	6	26	49			6	30	18	45			15	25	27			28	18	25		
7	7	27	50			7	31	19	46			21	29	36			31	22	26		
8	8	28				8	32	33	47			22	30	37			33	24	30		
9	9	29				21		34	51			23	40	42			48	29	36		
10	10	30				22		35	52			33	41	43				32	39		
11	11	31				23		36	53			34	44	46				34	41		
12	12	32				24		37	54			35	45	47				38	42		
13	13	33				25		38	55			48	51	53				40	45		
14	14	34				26		39	56			49	52	54				44	46		
15	15	35				27			57			50	55	57				49	52		
16	16	36				28			58				56	58				51	53		
17	17	37																55	56		
18	18	38																	57		
19	19	39																			
20	20	40																			
		41																			
		42																			
		43																			

Tabla 4.11 Listado de nodos visitados por los caminos desde 1 hasta 24 en G de 27 nodos

Nodos visitados por todos los caminos desde el fuente hasta el sumidero

Caminos	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23
1	1	19	42	50	58	1	8	13	37	55	56	1	3	5	41		1	2	4	6		
2	2	20	43	51	59	2	9	14	38	63	64	2	4	6	46		7	3	5	16		
3	3	21	44	52	60	3	10	15	39			13	8	15	54		10	8	9	24		
4	4	22	45	53	61	4	11	16	40			14	9	16	55		13	11	15	34		
5	5	23	46	54	62	5	12	17	41			19	21	23	56		19	14	18	40		
6	6	24	47	55	63	6	26	18	42			20	22	24	57		25	17	22	41		
7	7	25	48	56	64	7	27	31	43			31	26	33	62		28	20	23	45		
8	8	26	49	57		19	28	32	44			32	27	34	63		31	21	27	46		
9	9	27				20	29	33	45			47	37	39	64		47	26	33	49		
10	10	28				21	30	34	46			48	38	40				29	36	53		
11	11	29				22		35	50				42	44				32	38	54		
12	12	30				23		36	51				43	45				35	39	55		
13	13	31				24			52				50	52				37	43	56		
14	14	32				25			53				51	53				42	44	57		
15	15	33							54				58	60				48	51	61		
16	16	34							58				59	61				50	52	62		

17	35
18	36
	37
	38
	39
	40
	41

59
60
61
62

58	59	63
	60	64

Tabla 4.12 Listado de nodos visitados por los caminos desde 1 hasta 25 en G de 27 nodos

Nodos visitados por todos los caminos desde el fuente hasta el sumidero

Caminos	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**	**15**	**16**	**17**	**18**	**19**	**20**	**21**	**22**	**23**	**24**
1	1	12	29	39	57	1	6	8	23	45	49	1	2	3	26	36	1		2	4	27		
2	2	13	30	40	58	2	7	9	24	46	50	8	6	4	27	37	5		3	10	33		
3	3	14	31	41	59	3	17	10	25	47	51	12	13	9	28	52	7		6	15	36		
4	4	15	32	42	60	4	18	11	26	48	52	19	17	10	32	53	8		9	21	43		
5	5	16	33	43	61	5		19	27	63	53	35	23	14	33	70	12		11	25	46		
6	6	17	34	44	62	12		20	28	64	67		29	15	34	71	16		13	26	48		
7	7	18	35	45	63	13		21	29	65	68		39	20	42	72	18		14	31	50		
8	8	19	36	46	64	14		22	30	66	69		57	21	43	73	19		17	32	52		
9	9	20	37	47	65	15			31		70			24	44		35		20	38	55		
10	10	21	38	48	66	16			32		71			25	45				22	41	61		
11	11	22		49	67				33					30	46				23	42	64		
		23		50	68				34					31	47				24	45	66		
		24		51	69				39					40	49				29	49	68		
		25		52	70				40					41	50				30	54	70		
		26		53	71				41					58	51				39	59	72		
		27		54	72				42					59	54				40	60			
		28		55	73				43						55				57	63			
				56					44						56				58	67			
									57						60								
									58						61								
									59						62								
									60						63								
									61						64								
									62						65								
															67								
															68								
															69								

Tabla 4.13 Listado de nodos visitados por los caminos desde 1 hasta 26 en G de 27 nodos

Nodos visitados por todos los caminos desde el fuente hasta el sumidero

Caminos	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25
1	1	7	18	25	38	1	4	5	13	30	33		1	2	16	23				2	17			
2	2	8	19	26	39	2	10	6	14	31	34		4	3	17	35				5	22			
3	3	9	20	27	40	3		11	15	32	35		7	5	21	48				8	23			
4	4	10	21	28	41	7		12	16	43	46		10	6	22	49				11	29			
5	5	11	22	29	42	8			17	44	47		13	8	28					14	31			
6	6	12	23	30	43	9			18	45	48		18	9	29					16	32			
		13	24	31	44				19				25	11	30					19	34			
		14		32	45				20				38	12	31					21	35			
		15		33	46				21					14	33					24	37			
		16		34	47				22					15	34					26	42			
		17		35	48				25					19	36					28	44			
				36	49				26					20	37					30	45			
				37					27					26	41					33	47			
									28					27	42					36	48			
									29					39	43					39	49			
									38					40	44					41				
									39						46					43				
									40						47					46				
									41															
									42															

Tabla 4.14 Listado de nodos visitados por los caminos desde 1 hasta 27 en G de 27 nodos

Nodos visitados por todos los caminos desde el fuente hasta el sumidero

Caminos	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27
1	1	3	9	16	33	1		2	5	20	24			1	6	13				1	7					
2	2	4	10	17	34	3		4	6	21	25			2	7	14				2	11					
		5	11	18	35				7	22	26			3	8	27				3	13					
		6	12	19	36				8	23	27			4	10	28				4	18					
		7	13	20	37				9	37	28			5	11	44				5	21					
		8	14	21	38				10	38	29			9	12	45				6	23					
			15	22	39				11	39	41			16	17	47				9	25					
				23	40				12	40	42			33	18	48				10	27					
				24	41				16		43				19					15	31					
				25	42				17		44				20					16	35					
				26	43				18		45				21					17	38					
				27	44				19		46				22					20	40					
				28	45				33						24					24	42					
				29	46				34						25					30	44					

30	47
31	48
32	

35
36

26
30
31
32
34
35
36
37
38
39
41
42
43

33	47
34	
37	
41	

4.1.2.5 Cálculo de colisiones para cada nodo en el camino

Si por un nodo i de G pasa más de un camino desde s a t, se considera que en el nodo i hay colisión de caminos hacia el mismo sumidero. La cantidad de caminos que pasen por i, se considera su número de colisiones. El cálculo de las longitudes de cada $lista$ en cada $nodosenruta(i)$, determina el número de colisiones para cada nodo en los caminos del flujo desde el nodo s hasta el nodo t. La longitud (número de colisiones) se almacena en el arreglo $colisiones$ y, por ende, el tamaño de este arreglo es $t-2$. Cada $colisiones(i)$ significa el número de caminos que pasan por el nodo i desde el nodo etiquetado por 2, después de la fuente s, hasta el nodo previo al sumidero t, o sea $t-1$ (Aquí se tiene en cuenta que los nodos están ordenados topológicamente).

Ejemplo 4.6: Para el ejemplo del grafo G con 27 nodos, se muestran los vectores de colisiones para cada lista de caminos de cada nodo sumidero en la Tabla 4.15.

Tabla 4.15 Vectores de colisiones para cada lista de caminos de cada sumidero en G de 27 nodos

t	$colisiones$
23	[20,23,7,4,4,16,8,14,16,0,0,15,16,16,0,0,9,17,18,0,0]
24	[18,23,8,8,7,14,10,12,20,2,2,10,16,16,9,0,9,17,18,18,0,0]
25	[11,17,10,18,17,10,4,8,24,8,10,5,8,16,27,8,9,0,18,18,15,0,0]
26	[6,11,7,13,12,6,2,4,20,6,6,0,8,16,18,4,0,0,0,18,15,0,0,0]
27	[2,6,7,17,16,2,0,2,16,8,12,0,0,8,27,8,0,0,0,18,15,0,0,0,0]

Los nodos sin caminos que los atraviesen, tendrán una $lista$ vacía y, por ende, el número de colisiones es 0.

4.1.2.6 Cálculo de colisiones en cada camino

A partir del vector de $colisiones$, determinado en el paso anterior, se calcula la sumatoria de colisiones que se presentan en cada uno de los caminos hallados y, que son almacenados en $rutas$. Utilizando como índices los nodos del i-ésimo vector $camino$ en el arreglo $rutas$, se calcula la sumatoria de $colisiones$ que cada nodo, individualmente, aporta al total de colisiones del i-ésimo $camino$ en el arreglo $rutas$. Las colisiones para el i-ésimo $camino$ en $rutas$ se calculan según:

$$rutas(i).sumcol = \sum_{j=2}^{|long(i)|-1} colisiones(rutas(i).camino(j)) \tag{4.3}$$

Esta sumatoria selecciona y suma, del vector $colisiones$, el número de colisiones individuales para cada una de las posiciones o nodos $rutas(i).camino(j)$ con el fin de obtener, al final, la suma de colisiones $sumcol$ que se presentan en el i-ésimo $camino$ de $rutas$.

Ejemplo 4.7: A partir de la Tabla 4.16 hasta la Tabla 4.20, se muestra el número total de colisiones para el i-ésimo $camino$ en $rutas$ de cada sumidero $t \in \{23, 24, 25, 26, 27\}$ en el grafo de 27 nodos.

Tabla 4.16 Sumatoria de colisiones en cada camino desde 1 hasta 23 en G de 27 nodos

i	$rutas(i).sumcol$	i	$rutas(i).sumcol$	i	$rutas(i).sumcol$	i	$rutas(i).sumcol$
1	60	15	49	29	64	43	55
2	68	16	68	30	65	44	56
3	51	17	50	31	40	45	57
4	69	18	51	32	48	46	57
5	70	19	52	33	61	47	39
6	70	20	20	34	69	48	31
7	52	21	63	35	52	49	39
8	45	22	71	36	71	50	22
9	61	23	54	37	53	51	53
10	62	24	72	38	54	52	54
11	37	25	73	39	55	53	54
12	45	26	73	40	72	54	36
13	58	27	55	41	73	55	53
14	66	28	48	42	73	56	54
						57	54
						58	36

Tabla 4.17 Sumatoria de colisiones en cada camino desde 1 hasta 24 en G de 27 nodos

i	$rutas(i).sumcol$	i	$rutas(i).sumcol$	i	$rutas(i).sumcol$	i	$rutas(i).sumcol$
1	51	17	47	33	69	49	26
2	59	18	48	34	69	50	61
3	65	19	56	35	52	51	62
4	66	20	64	36	53	52	62
5	66	21	70	37	76	53	62
6	66	22	71	38	77	54	55
7	41	23	71	39	77	55	37
8	61	24	71	40	77	56	37
9	62	25	46	41	70	57	35

10	37	26	66	42	61	58	60
11	45	27	67	43	62	59	61
12	28	28	42	44	62	60	61
13	49	29	50	45	62	61	61
14	57	30	33	46	55	62	54
15	64	31	54	47	27	63	36
16	64	32	62	48	35	64	36

Tabla 4.18 Sumatoria de colisiones en cada camino desde 1 hasta 25 en G de 27 nodos

i	$rutas(i).sumcol$	i	$rutas(i).sumcol$	i	$rutas(i).sumcol$	i	$rutas(i).sumcol$
1	35	19	39	37	18	55	60
2	47	20	59	38	28	56	45
3	55	21	59	39	68	57	67
4	55	22	43	40	76	58	75
5	30	23	67	41	76	59	75
6	41	24	75	42	87	60	86
7	24	25	75	43	84	61	83
8	33	26	86	44	69	62	68
9	53	27	83	45	71	63	70
10	53	28	68	46	68	64	67
11	37	29	60	47	53	65	52
12	41	30	68	48	41	66	40
13	53	31	68	49	73	67	72
14	61	32	79	50	70	68	69
15	61	33	76	51	55	69	54
16	36	34	61	52	51	70	50
17	47	35	24	53	36	71	35
18	30	36	33	54	63	72	40
						73	25

Tabla 4.19 Sumatoria de colisiones en cada camino desde 1 hasta 26 en G de 27 nodos

i	$rutas(i).sumcol$	i	$rutas(i).sumcol$	i	$rutas(i).sumcol$	i	$rutas(i).sumcol$
1	20	13	39	25	41	37	46
2	46	14	65	26	67	38	40
3	28	15	47	27	49	39	66
4	16	16	67	28	69	40	48
5	44	17	64	29	66	41	68
6	26	18	35	30	55	42	65
7	25	19	61	31	52	43	54

8	51	20	43	32	34	44	51
9	33	21	63	33	55	45	33
10	21	22	60	34	52	46	54
11	49	23	26	35	38	47	51
12	31	24	25	36	49	48	37
						49	31

Tabla 4.20 Sumatoria de colisiones en cada camino desde 1 hasta 27 en G de 27 nodos

i	$rutas(i).sumcol$	i	$rutas(i).sumcol$	i	$rutas(i).sumcol$	i	$rutas(i).sumcol$
1	30	13	30	25	71	37	69
2	30	14	15	26	56	38	66
3	34	15	25	27	52	39	51
4	34	16	59	28	37	40	39
5	48	17	78	29	29	41	73
6	67	18	75	30	62	42	70
7	64	19	60	31	59	43	55
8	49	20	70	32	44	44	51
9	49	21	67	33	58	45	36
10	68	22	52	34	77	46	28
11	65	23	40	35	74	47	39
12	50	24	74	36	59	48	24

4.1.2.7 Búsqueda de los caminos primarios

Se inicializa el arreglo $marca$ de longitud $nrutas$ con valores en cero para la búsqueda de los caminos que conforman el flujo hacia un nodo sumidero t. Es decir, $marca(i) = 0, \forall i = 1, \dots, nrutas$. Luego se ejecutan los siguientes pasos mientras haya un camino i con $marca(i) = 0$.

a. Búsqueda de camino sin marcar de menor longitud: Se busca entre todos los caminos marcados con 0, el de menor longitud (se busca el mínimo valor en el arreglo $long$ entre las posiciones que hayan sido marcadas con 0). Los caminos de menor longitud tendrán menos probabilidad de presentar nodos comunes, en un paso futuro, al momento de mezclarse con los caminos que conducen a otros sumideros. Este valor mínimo de longitud o del camino más corto para alcanzar al sumidero t, se guarda en la variable mn. Es decir,

$$mn = \min\{long(i) | marca(i) = 0, \forall i = 1, \dots, nrutas\} \quad (4.4)$$

Ejemplo 4.8: Aplicando (4.4) al ejemplo de G de 27 nodos para el nodo sumidero 23 en la primera iteración, la mínima longitud de camino obtenida de long es $mn = 3$, como se observa en la primera fila de la Tabla 4.9.

b. Almacenamiento de identificadores y longitudes de caminos sin marcar: Se almacena para cada $camino\ i$ sin marcar ($marca(i) = 0$) o que no ha sido revisado, su identificador en el arreglo pos y su longitud en el arreglo ln. Los identificadores serán utilizados para hallar el camino

más corto y con menos colisiones. Las longitudes de caminos en ln, es un subconjunto del arreglo $long$ ($ln \subseteq long$). En cada iteración del algoritmo serán menos los identificadores de caminos sin marcar. El objetivo de este paso es que si $marca(i) = 0$, entonces i se incluye en pos, y $long(i)$ se incluye en ln, según las expresiones (4.5) y (4.6):

$$pos = [i_1, i_2, \dots, i_k], \text{ tal que } marca(i_j) = 0, 1 \leq i_j \leq nrutas \quad (4.5)$$

$$ln = [long(i_1), long(i_2), \dots, long(i_k)], \text{ tal que } marca(i_j) = 0, 1 \leq i_j \leq nrutas \quad (4.6)$$

El tamaño de los arreglos pos y ln es el número de identificadores de caminos sin marcar y se denotará por $nrutsm$. Se observa esta búsqueda en el Algoritmo 4.2.

Ejemplo 4.9: Según el ejemplo de G de 27 nodos, los vectores pos y ln para el sumidero $t = 23$ en la primera iteración, comprenden todos los identificadores de los $caminos\ i$ y todas las longitudes en $long$, respectivamente. Es decir, ln corresponde a la primera fila de la Tabla 4.9 y $pos = [1, \dots, 58]$.

c. Búsqueda de los caminos no marcados con menor longitud: Se lleva a cabo la búsqueda de los identificadores de camino en ln que coinciden con la mínima longitud de camino mn. Estos identificadores se guardan en el arreglo rut. Es decir, según la expresión (4.7):

$$rut = [i_1, i_2, \dots, i_k], \text{ tal que } ln(l) = mn, pos(l) = i_j, 1 \leq l \leq nrutsm \quad (4.7)$$

Ejemplo 4.10: Según el Ejemplo 4.9, el arreglo de identificadores de caminos para la primera iteración cuando el sumidero es 23, corresponde con $rut = [20]$. Esto quiere decir, que la posición 20 es el único identificador de $caminos$ con la mínima longitud de 3.

Algoritmo 4.2: Búsqueda_Identificadores_Longitudes_Caminos_Sin_Marcar

Entrada: Arreglos $marca, long$, Escalar $nrutas$
Salida: Arreglos pos, ln

```
15 Para i = 1:nrutas
16     k = 1;
17     Si marca(i) == 0
18         pos(k) = i;
19         ln(k) = long(i);
20         k++;
21     Fin Si
22 Fin Para
23 Retornar pos, ln;
```

d. Selección del camino con menos colisiones y hallazgo del valor mínimo de colisiones: De entre todos los caminos registradas en rut, se escoge el de menor número de colisiones. En la variable $prut$ se guarda la posición de rut en que está almacenado el identificador de caminos con menos colisiones. Es decir, $rut(prut)$ es el camino que tendrá menos colisiones entre los caminos que presentan la mínima longitud. A este camino se le denominará el *camino*

primario. A través de esta búsqueda, también se halla el valor correspondiente al mínimo de colisiones que se almacena en la variable $mncol$. Formalmente, el cálculo del mínimo de colisiones y la posición $prut$ del identificador de camino con menor número de colisiones, corresponde con las expresiones en (4.8) y (4.9):

$$mncol = \min\{rutas(rut(i)).sumcol | 1 \leq i \leq nrutsm\} \tag{4.8}$$

$$prut = i, \text{tal que } rutas(rut(i)).sumcol = mncol \tag{4.9}$$

Ejemplo 4.11: Para el Ejemplo 4.10 del grafo G de 27 nodos con nodo sumidero 23, en la primera iteración, el $mncol = 20$, que corresponde con el camino cuyo identificador $rut(prut) = 20$ con $prut = 1$.

e. Conteo de segundos nodos: El número de caminos máximos posibles para la comunicación entre el nodo fuente s y el sumidero $t_i, i = 1, \dots, |T|$, está almacenado en $flumaxrut(i)$. Cada uno de las $flumaxrut(i)$ caminos primarios para establecer el flujo hacia el sumidero t_i tiene la forma $[s, b_{2j}, a_2, \dots, a_\ell, t_i]$, y cada uno pasa por un segundo nodo denotado por $b_{2j}, j = 1, \dots, flumaxrut(i)$ con $b_{2j} \neq b_{2k}, j \neq k$. Es decir, los $flumaxrut(i)$ caminos, desde el nodo fuente s hasta cada nodo sumidero t_i, son disyuntos y, por ende, cada uno de los caminos utilizan un segundo nodo distinto ya que no hay nodos repetidos entre caminos diferentes. Se establece el segundo nodo b_{2j} como un elemento distintivo para cada camino en un flujo desde el nodo fuente s y un nodo sumidero t_i. Con la búsqueda de los mejores caminos desde el nodo fuente s hasta el nodo sumidero t_i, se establece el número de veces que es utilizado cada segundo nodo en todos los caminos para el total de nodos sumideros $|T|$. Este conteo se va actualizando en el arreglo $contsegnod$, donde cada $contsegnod(j)$ es el número de veces que el $segnodtot(j)$ es utilizado en todos los caminos que conforman los flujos hacia los $|T|$ sumideros. Es claro, que después de llevar a cabo todas las iteraciones para hallar los flujos hacia los nodos sumideros, se puede establecer la relación (4.10).

$$\sum_{i=1}^{|T|} flumaxrut(i) = \sum_{j=1}^{|contsegnod|} contsegnod(j) \tag{4.10}$$

Ejemplo 4.12: En la Tabla 4.21 y, de acuerdo con el grafo G de 27 nodos, se muestra la secuencia de cálculos de $contsegnod$ y $flumaxrut$, según se van realizando las iteraciones sobre nd, el índice de los nodos sumideros. Se confirma la relación (4.10).

Tabla 4.21 Secuencia de hallazgos de $contsegnod$ y $flumaxrut$ para G de 27 nodos

nd	$contsegnod$	$flumaxrut$
1	[1,1,1,1,0]	[4]
2	[2,2,2,2,0]	[4,4]
3	[3,3,3,3,1]	[4,4,5]
4	[4,4,4,3,2]	[4,4,5,4]
5	[5,4,5,4,3]	[4,4,5,4,4]

f. Actualización del arreglo $marca$ y eliminación de $caminos$ que colisionan con el camino primario hallado: Seleccionado el camino primario, se busca en cada $lista$ registrada en

$nodosenruta$, cuáles colisionan con el *camino primario* identificado por $rut(prut)$. Es decir, se busca en cada $lista$ determinada por $nodosenruta$, si el *camino primario* $rut(prut)$ está allí; luego los identificadores de $caminos$ distintos del *camino primario*, deben ser eliminados para que no colisionen con este.

El procedimiento descrito en el Algoritmo 4.3, consiste en que una vez encontrada una $lista$ de $nodosenruta$ que contenga el *camino primario* $rut(prut)$, se incluyan todos los identificadores de los caminos distintos del *primario* en una pila de eliminación $pilaelim$.

Algoritmo 4.3: Caminos_Eliminar

Entrada: Arreglos $rut, nodosenruta$, Escalar $prut$
Salida: Arreglos $pilaelim$

```
1  pilaelim = ∅;
2  Para j = 1:|nodosenruta|
3     Si rut(prut) ∈ nodosenruta(j).lista
4        Insertar en pilaelim a {k|k ∈ nodosenruta(j).lista\rut(prut)};
5     Fin Si
6  Fin Para
7  Eliminar los elementos duplicados de pilaelim;
8  Retornar pilaelim;
```

Posteriormente, según (4.11), se marcan todas las posiciones del arreglo $marca$ correspondientes a los identificadores en $pilaelim$ con valor en 1, que significa que estos caminos no serán utilizados en el flujo hacia el sumidero t. Además, de acuerdo con (4.12), se anulan, en el arreglo $rutas$, los $caminos$ correspondientes a los identificadores en $pilaelim$, para que en el futuro ya no vuelvan a utilizarse en la marcación de nodos y, en consecuencia, no aumentar el número de colisiones en las siguientes iteraciones.

$$marca(i) = 0, \forall i \in pilaelim \tag{4.11}$$

$$rutas(i).camino = \emptyset, \forall i \in pilaelim \tag{4.12}$$

Seguidamente, según (4.13), se busca en cada $lista$ de $nodosenruta$, los identificadores de $caminos$ que coincidan con los ubicados en $pilaelim$. Estos identificadores de $caminos$ se eliminan de cada $lista$ en $nodosenruta$ y, así se podrá disminuir el número de colisiones para la siguiente iteración.

$$nodosenruta(j).lista(k) = \emptyset, \forall k \in pilaelim \cap nodosenruta(j).lista, \quad \forall j = 1, \dots, |nodosenruta| \tag{4.13}$$

Ejemplo 4.13: De acuerdo con lo explicado en este paso, los caminos a eliminar en la primera iteración para el hallazgo del flujo máximo desde 1 hasta 23 en el grafo G de 27 nodos, comprenden desde el identificador 1 al 19, los cuales son distintos del camino primario 20. Esto genera una actualización de cada $camino$ en el arreglo $rutas$, correspondientes al sumidero 23 (Tabla 4.22) y cada $lista$ en el arreglo $nodosenrutas$ (Tabla 4.23).

g. Re-cálculo de colisiones y suma de colisiones: Se re-totalizan las colisiones que generan los distintos caminos restantes desde el nodo fuente s hasta el nodo sumidero $t = dest(nd)$ para cada uno de los nodos que conforman estos caminos. Después se vuelven a calcular las sumatorias de colisiones para cada uno de los caminos restantes y se actualizan estos valores.

Tabla 4.22 Actualización de caminos desde el nodo fuente 1 al sumidero 23 en G de 27 nodos

i	$rutas(i).camino$	i	$rutas(i).camino$	i	$rutas(i).camino$	i	$rutas(i).camino$
1		15		29	1 3 8 14 19 23	43	1 3 10 15 23
2		16		30	1 3 8 14 20 23	44	1 4 10 14 19 23
3		17		31	1 3 8 18 23	45	1 4 10 14 20 23
4		18		32	1 3 8 19 23	46	1 4 10 15 20 23
5		19		33	1 3 9 13 18 23	47	1 4 10 15 23
6		20	1 2 23	34	1 3 9 13 19 23	48	1 4 13 18 23
7		21	1 3 7 13 18 23	35	1 3 9 13 23	49	1 4 13 19 23
8		22	1 3 7 13 19 23	36	1 3 9 15 20 23	50	1 4 13 23
9		23	1 3 7 13 23	37	1 3 9 15 23	51	1 5 10 14 19 23
10		24	1 3 7 14 19 23	38	1 3 9 19 23	52	1 5 10 14 20 23
11		25	1 3 7 14 20 23	39	1 3 9 20 23	53	1 5 10 15 20 23
12		26	1 3 7 15 20 23	40	1 3 10 14 19 23	54	1 5 10 15 23
13		27	1 3 7 15 23	41	1 3 10 14 20 23	55	1 6 10 14 19 23
14		28	1 3 7 18 23	42	1 3 10 15 20 23	56	1 6 10 14 20 23
						57	1 6 10 15 20 23
						58	1 6 10 15 23

Tabla 4.23 Actualización de nodos visitados por los caminos desde 1 hasta 23 en G de 27 nodos

Nodos visitados por todos los caminos desde el fuente hasta el sumidero

Caminos	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**	**15**	**16**	**17**	**18**	**19**	**20**	**21**	**22**
		21	44	51	55				40												
		22	45	52	56				41												
		23	46	53	57				42												
		24	47	54	58				43												
		25	48				29		44				24	26			21				
		26	49				30		45				25	27			28		25		
		27	50				31		46			21	29	36			31	22	26		
		28					32	33	47			22	30	37			33	24	30		
		29				21		34	51			23	40	42			48	29	36		
		30				22		35	52			33	41	43				32	39		
		31				23		36	53			34	44	46				34	41		
		32				24		37	54			35	45	47				38	42		
		33				25		38	55			48	51	53				40	45		
		34				26		39	56			49	52	54				44	46		
		35				27			57			50	55	57				49	52		
		36				28			58				56	58				51	53		
		37																55	56		
		38																	57		
		39																			

20	40
	41
	42
	43

Ejemplo 4.14: La Tabla 4.24 muestra la actualización de las colisiones de cada camino que finaliza en el nodo sumidero 23 para el grafo G de 27 de nodos, y la Tabla 4.25 muestra la actualización de la sumatoria de colisiones para estos mismos caminos.

Tabla 4.24 Actualización de colisiones para cada camino hacia el sumidero 23 en G de 27 nodos

t	$colisiones$
23	[0,1, 23,7,4,4,8,4,7,16,0,0,9,12,12,0,0,5,11,13,0,0]

Tabla 4.25 Actualización sumatoria de colisiones en cada camino desde 1 hasta 23 en G de 27 nodos

i	$rutas(i).sumcol$	i	$rutas(i).sumcol$	i	$rutas(i).sumcol$	i	$rutas(i).sumcol$
1	0	15	0	29	50	43	51
2	0	16	0	30	52	44	46
3	0	17	0	31	32	45	48
4	0	18	0	32	38	46	48
5	0	19	0	33	44	47	35
6	0	20	1	34	50	48	21
7	0	21	45	35	39	49	27
8	0	22	51	36	55	50	16
9	0	23	40	37	42	51	43
10	0	24	54	38	41	52	45
11	0	25	56	39	43	53	45
12	0	26	56	40	62	54	32
13	0	27	43	41	64	55	43
14	0	28	36	42	64	56	45
						57	45
						58	32

h. Fijación de camino primario: El identificador de camino primario ubicado en $rut(prut)$ se marca como fijo con valor de 2 en $marca$. Es decir, se asigna $marca(rut(prut)) = 2$.

i. Resumen de los caminos válidos: Se calcula el identificador de caminos válidos, y cada uno de estos se guarda en el arreglo $rutasval$. Estos identificadores se obtienen a partir del arreglo $marca$, donde se seleccionan las posiciones cuyo contenido es 2, y estas posiciones corresponden con los identificadores de los caminos válidos. Esto quiere decir que si $marca(i) = 2$, el camino i hace parte de los caminos válidos desde s hasta t.

En la estructura vectorial $rutasdest$ (tiene una posición por cada sumidero), se va almacenando internamente otra estructura vectorial denominada $rutamax$, la cual contiene el

campo $camino$, que corresponde al arreglo formado por cada camino válido seleccionado; y $numrutas$, que equivale al identificador de $camino$ (índice del arreglo $rutas$) escogido. El formato de esta estructura vectorial es:

$rutasdest(nd)$
  $rutamax(i)$
    $camino$
    $numruta$

El arreglo vectorial $segnod$, de tamaño igual al número de sumideros, guarda el segundo nodo de cada $camino$ válido en el arreglo $lista$. Luego, $segnod(nd).lista(i) = seg$, significa que el segundo nodo seg está en el camino i que llega a t_{nd}. El formato de esta estructura vectorial es:

$segnod(nd)$
  $lista$

En el arreglo $flumaxrut$, se almacena la cantidad de caminos disyuntos obtenidos por cada nodo sumidero. Al final de todas las iteraciones, se obtendrá en este arreglo, el flujo máximo por cada sumidero.

Ejemplo 4.15: En la Tabla 4.26 se observa el resumen de identificadores de caminos válidos hacia cada sumidero; además, se presentan los segundos nodos que los identifican. En la misma tabla se encuentran el identificador de camino que le corresponde según cada sumidero y la secuencia de nodos que conforman cada camino. En la Figura 4.2 se muestran los grafos de flujo máximo correspondientes a cada uno de los sumideros presentados en la Tabla 4.26. El flujo máximo que llega al nodo sumidero 25 (Figura 4.2 (c)), es el único que sobrepasa el valor del mínimo flujo máximo común, que es 4.

Tabla 4.26 Identificación y descripción de caminos por cada sumidero

nd	t_{nd}	$rutasval$	$segnod(nd).lista$	$rutasdest$		
				i	$rutamax$	
					$numruta$	$camino$
1	23	[20,31,50,54]	[2,3,4,5]	1	20	[1 2 23]
				2	31	[1 3 8 18 23]
				3	50	[1 4 13 23]
				4	54	[1 5 10 15 23]
				i	$rutamax$	
					$numruta$	$camino$
2	24	[12,25,49,50]	[2,3,4,5]	1	12	[1 2 8 24]
				2	25	[1 3 7 18 24]
				3	49	[1 4 21 24]
				4	50	[1 5 10 14 19 24]
				i	$rutamax$	
					$numruta$	$camino$
3	25	[7,22,37,56,66]	[2,3,4,5,6]	1	7	[1 2 8 18 25]
				2	22	[1 3 9 20 25]
				3	37	[1 4 17 25]
				4	56	[1 5 16 25]
				5	66	[1 6 11 22 25]
				i	$rutamax$	

4	26	[4,9,24,49]	[2,3,4,6]		***numruta***	***camino***
				1	4	[1 2 8 14 26]
				2	9	[1 3 7 15 26]
				3	24	[1 4 21 26]
				4	49	[1 6 17 22 26]
				i	***rutamax***	
					numruta	***camino***
5	27	[1,14,32,46]	[2,4,5,6]	1	1	[1 2 7 15 21 27]
				2	14	[1 4 17 27]
				3	32	[1 5 16 27]
				4	46	[1 6 12 27]

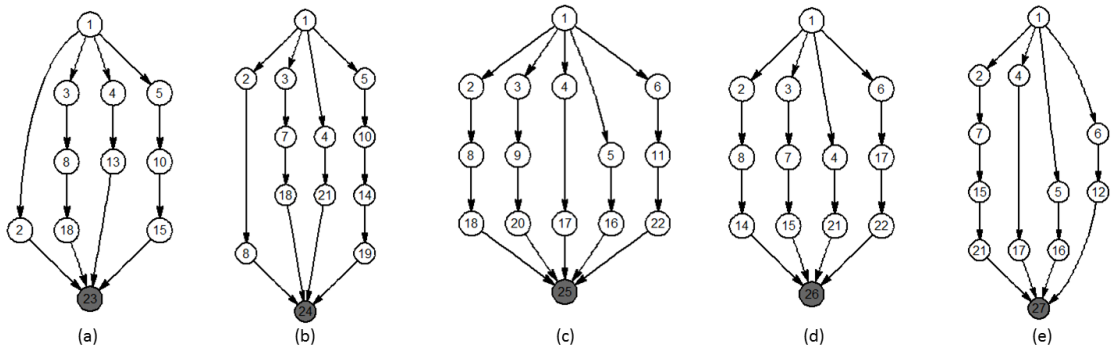


Figura 4.2 Grafos unicast de flujos máximos individuales para los sumideros de G de 27 nodos

4.2 Grafo de Mínimo Flujo Máximo

Después de obtener para cada sumidero el grafo de flujo máximo, se procede a unificar el sistema multicast unisesión al mínimo flujo máximo [5]. Esta sección explica cómo reducir la estructura vectorial $rutasdest$, de tal forma que en esta solo queden almacenados, por cada nodo sumidero que la constituyen, un número de caminos disyuntos que finalicen en el sumidero y que correspondan en cantidad al mínimo flujo máximo. En esta sección se calcula el grafo de mínimo flujo máximo $minmaxflujo$, que comprende todos los caminos desde el nodo fuente hasta cada sumidero. La cantidad de caminos por cada nodo sumidero es la misma y equivalente al mínimo flujo máximo.

4.2.1 Bases del algoritmo

Hasta este punto se han calculado los grafos unicast de flujos máximos G_i' individuales por cada nodo sumidero $t_i \in T$. Los grafos se clasifican en los que cumplen con el mínimo flujo máximo común r y los que no. Por cada grafo unicast de flujo máximo, ya está calculada la lista de segundos nodos $segnod(i).lista$, que identifican los caminos disyuntos que arriban al sumidero t_i.

El algoritmo unifica el número de caminos disyuntos que debe finalizar en los sumideros, de tal forma que todos tengan un flujo máximo igual a r. La política del algoritmo consiste en tomar cada G_i' que no cumpla con r ($f_{G_i'} > r$), verificar que la mayor cantidad de segundos nodos (en lo posible r) en $segnod(i).lista$, representantes de sus caminos, se solapen o intersecten con los segundos nodos de algunos de los grafos que si cumplan con el mínimo flujo r. Además, la sumatoria de las longitudes de los caminos hallados en G_i', debe ser la menor posible. Aquel conjunto de segundos nodos intersectados que se aproxime o sea igual a r, y que la longitud de los caminos que

representan en G_i', sea la menor posible, serán escogidos. Formalmente en (4.14), se calcula la intersección I_i^j de los segundos nodos de G_i' ($f_{G_i'} > r$) con los segundos nodos de G_j' ($f_{G_j'} = r$).

$$\mathrm{I}_i^j = segnod(i).lista \cap segnod(j).lista \tag{4.14}$$

Se define $\Lambda_{\mathrm{I}_i^j}$ como la suma de las longitudes de los caminos que contienen a los segundos nodos en I_i^j, la cual se calcula según (4.15).

$$\Lambda_{\mathrm{I}_i^j} = \sum_{k:rutasdest(i).rutamax(k).camino(2)\in \mathrm{I}_i^j} |rutasdest(i).rutamax(k).camino| \tag{4.15}$$

La colección de intersecciones para G_i', se denota como I_i y el conjunto de sumas de longitudes se denota como Λ_{I_i}. Estas colecciones se calculan según (4.20) y (4.21).

$$\mathrm{I}_i = \left\{\mathrm{I}_i^j \mid f_{G_i'} > r \ \wedge f_{G_j'} = r\right\} \tag{4.16}$$

$$\Lambda_{\mathrm{I}_i} = \left\{\Lambda_{\mathrm{I}_i^j} | \mathrm{I}_i^j \in \mathrm{I}_i\right\} \tag{4.17}$$

El conjunto intersección de segundos nodos se define según (4.22).

$$mincomun = \max\left\{\left|\mathrm{I}_i^j\right| | \mathrm{I}_i^j \in \mathrm{I}_i \ \wedge \Lambda_{\mathrm{I}_i^j} = \min(\Lambda_{\mathrm{I}_i})\right\} \tag{4.18}$$

En consecuencia, la selección del mejor conjunto intersección de segundos nodos no involucra a más de un grafo de mínimo flujo máximo ya obtenido. Es decir, los caminos son comunes con un solo grafo de mínimo flujo máximo, por ende no hay caminos entremezclados de varios grafos.

En caso de que no se alcance el flujo máximo común para un grafo, se debe seguir buscando entre todas sus rutas que no hayan sido utilizadas en la revisión, hasta completar el mínimo flujo máximo de caminos disyuntos con la menor sumatoria de longitudes. Los grafos que se modifiquen y obtengan su mínimo flujo máximo, pasan a la lista de los que cumplen.

Ejemplo 4.16: En la Figura 4.2(c), se observa que el grafo G_3' supera el mínimo flujo máximo $r = 4$. Se denotan los sumideros secuencialmente como $t_1 = 23, t_2 = 24, t_3 = 25, t_4 = 26$ y $t_5 = 27$. Para lograr llevar este grafo al mínimo flujo máximo común, se calcula, inicialmente, los conjuntos intersecciones según (4.14) y el conjunto de las sumas de longitudes que estos segundos nodos representan, según (4.15):

$\mathrm{I}_3^1 = \mathrm{I}_3^2 = \{2,3,4,5\}$, $\mathrm{I}_3^4 = \{2,3,4,6\}$ y $\mathrm{I}_3^5 = \{2,4,5,6\}$.
$\Lambda_{\mathrm{I}_3^1} = \Lambda_{\mathrm{I}_3^2} = 18$, $\Lambda_{\mathrm{I}_3^4} = 19$ y $\Lambda_{\mathrm{I}_3^5} = 18$.

Las colecciones de conjuntos intersecciones y sumas de longitudes de caminos son:
$\mathrm{I}_3 = \{\{2,3,4,5\},\{2,3,4,6\},\{2,4,5,6\}\}$, $\Lambda_{\mathrm{I}_3} = \{18,19,18\}$.

Por tanto, $mincomun = \{2,3,4,5\}$.

Esto significa, que los caminos de G_3' determinados por los segundos nodos 2,3, 4 y 5, son los escogidos para cumplir con el mínimo flujo máximo $r = 4$. El grafo G_3' resultante se muestra en la Figura 4.3.

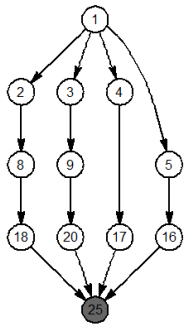


Figura 4.3 G_3' con flujo $r = 4$ igualado con otros grafos en G de 27 nodos

4.2.2 Cálculo del mínimo flujo máximo

A partir del arreglo $flumaxrut$, el cual contiene los valores de los flujos máximos desde el nodo fuente hacia cada uno de los nodos sumideros de la sesión multicast, se calcula el mínimo valor que corresponde con el mínimo flujo máximo de esta sesión. Este valor está almacenado en la variable $minflujo$ y se calcula según (4.19).

$$minflujo = \min\{flumaxrut(i) | 1 \leq i \leq ndest\} \tag{4.19}$$

En el arreglo $dirflmin$, se almacenan los índices de $flumaxrut$ que corresponden con el $minflujo$. Estos índices, igualmente, corresponden con los de $rutasdest$ que coinciden con el mínimo flujo máximo.

Ejemplo 4.17: Según los datos de la Tabla 4.21, el mínimo flujo máximo $minflujo$ es 4 y el arreglo $dirflmin$ contiene los índices donde $flumaxrut$ coincide con $minflujo$; es decir, $dirflmin = [1,2,4,5]$.

4.2.3 Emparejamiento de caminos que no cumplen

En el arreglo $dirflrut$ se almacenan los índices de $flumaxrut$, cuyos flujos máximos son mayores que el $minflujo$, según se especifica en (4.20). Los índices almacenados en $dirflrut$ coinciden con los índices de los sumideros en $rutasdest$ que sobrepasan en caminos entrantes al mínimo flujo máximo, según (4.21). Si este arreglo está vacío, significa que todos los caminos hacia los sumideros, son iguales al $minflujo$, y no hay que establecer ajustes. Cuando no está vacío, el procedimiento de ajuste consiste en reacomodar las rutas.

$$i \in dirflrut \Leftrightarrow flumaxrut(i) > minflujo \tag{4.20}$$

$$\forall i \in dirflrut \text{ y } |rutasdest(i).rutamax| > minflujo \tag{4.21}$$

El objetivo es lograr que el número de $caminos$ hacia los sumideros, con índices en $rutasdest$, y almacenados en $dirflrut$, sea igual en cantidad al $minflujo$. La meta se logra con la selección de los caminos que tengan en común, con los flujos de los sumideros con índice en $dirflmin$, el segundo nodo que los determina y que la sumatoria de sus longitudes sea la menor. Esta tarea se realiza con la ejecución del Algoritmo 4.4. Este algoritmo implica la ejecución de un conjunto de

iteraciones según el número de índices de sumideros almacenados en $dirflrut$. Dentro de cada iteración se ejecutan los siguientes procedimientos:

Algoritmo 4.4: Igualar_Sumideros_Mayores_Flujo

Entrada: Arreglos $rutasdest, segnod, dirflrut, dirflmin$ Escalar $minflujo$
Salida: Arreglo $rutasdest, segnod, dirflmin$

```
1  Para cada sumidero i en dirflrut
2     Búsqueda_Sobre_Nodos_Comunes(rutasdest, segnod, dirflmin, i, minflujo);
3     Búsqueda_Sobre_Nodos_Comtotal(minconun, comtotal, segnod, longitud, i,
                                    minflujo, minlong);
4     Actualizaciones(rutasdest, segnod, dirflmin, mincomun, comtotal, longitud, i,
                      minflujo);
5  Fin Para
6  Retornar rutasdest, segnod, dirflmin;
```

4.2.3.1 Búsqueda sobre nodos comunes

El Algoritmo 4.5 calcula, para el sumidero i en $dirflrut$ entregado como parámetro, y para cada sumidero j en $dirflmin$, los segundos nodos comunes pertenecientes al conjunto de caminos que convergen en i y al conjunto de caminos que convergen en j. Es decir, establece una búsqueda de segundos nodos comunes entre el sumidero i con cada sumidero j, cuyo grafo de flujo máximo G_j' si cumple el $minflujo$. Además, la sumatoria de las longitudes de los caminos, identificados por los segundos nodos comunes que finalizan en i, debe ser la mínima de entre todas las $|dirflmin|$ posibilidades.

De todos los conjuntos obtenidos, se calcula el mínimo de segundos nodos comunes ($mincomun$) con la condición que se logre alcanzar el $minflujo$ y que la sumatoria de las longitudes de los caminos que estos segundos nodos determinan, sea la mínima posible ($minlong$). Si no se logra alcanzar un mínimo común de segundos nodos igual al $minflujo$, se devuelve el arreglo mínimo común más cercano posible en cantidad a este límite, y con la sumatoria de longitudes que sea la menor posible. Siempre será preferido un conjunto de segundos nodos comunes igual en cantidad al $minflujo$, sobre uno que no lo sea. Durante cada iteración sobre los sumideros en $dirflmin$, se van almacenando, de forma única, todos los segundos nodos comunes en el arreglo $comtotal$, el cual es una salida del algoritmo que será utilizada para verificar si se encuentra un $mincomun$ de tamaño $minflujo$ con una menor sumatoria de longitudes en los caminos que estos representan. Es decir, todavía es posible hallar un $mincomun$ mejor que el encontrado, usando el arreglo $comtotal$, el cual contiene los segundos nodos de todos los sumideros en $dirflmin$ validados. Este algoritmo también devuelve la $longitud$ de los caminos que finalizan en el sumidero en la posición i de $dirflrut$; es decir, la longitud de los caminos almacenados en $rutasdest(i)$. El arreglo $longitud$ se calcula según la expresión (4.22). En conclusión, este algoritmo devuelve un arreglo de segundos nodos que identifican los posibles caminos hacia el nodo sumidero asociado a i en $dirflrut$. Este arreglo, no sobrepasa el minimo flujo máximo $minflujo$.

$$longitud = \left\{ \begin{matrix} |rutasdest(i).rutamax(k).camino| \; | \\ i \in dirflrut, 1 \leq k \leq |rutasdest(i).rutamax| \end{matrix} \right\} \tag{4.22}$$

El cálculo de los segundos nodos comunes se ejecuta $|dirflmin|$ veces en el Algoritmo 4.5, por tanto se ejecuta $|dirflrut| \times |dirflmin|$ veces dentro del Algoritmo 4.4.

Algoritmo 4.5: Búsqueda_Sobre_Nodos_Comunes

Entrada: Arreglos $rutasdest, segnod, dirflmin$ Escalar $i, minflujo$
Salida: Arreglos $mincomun, comtotal, longitud$ Escalar $minlong$

```
1  longitud = |rutasdest(i).rutamax(k).camino|, ∀ camino k finalizando en i;
2  minlong = ∞;
3  comtotal = ∅;
4  difcomun = |segnod(i).lista|;
5  mejorcomun = false;
6  mincomun = ∅;
7  Para cada sumidero j en dirflmin
8      comun = segnod(i).lista ∩ segnod(j).lista;
9      comtotal = comtotal ∪ comun;
10     nuevocomun = false;
11     Si |comun| == minflujo
12         comunsel = comun;
13         nuevocomun = true;
14         Si ~mejorcomun //Es el primer |comun| == minflujo
15             minlong = ∞;
16             mejorcomun = true;
17         Fin Si
18     Sino
19         Si ~mejorcomun //No hay un comun tal que |comun| == minflujo
20             Si |segnod(i).lista| − |comun| < difcomun
21                 difcomun = |segnod(i).lista| − |comun|;
22                 comunsel = comun;
23                 nuevocomun = true;
24             Fin Si
25         Fin Si
26     Fin Si
27     Si nuevocomun
28         sumlong = ∑_m longitud(m), tal que segnod(i).lista(m) == comunsel(k),
                     ∀k = 1, … , |comunsel|;
29     Fin Si
30     Si minlong > sumlong
31         minlong = sumlong;
32         mincomun = comunsel;
33     Fin Si
34 Fin Para
35 Retornar mincomun, comtotal, longitud, minlong;
```

4.2.3.2 Búsqueda sobre nodos comunes totales

Si el Algoritmo 4.5 genera un arreglo $mincomun$ de tamaño $minflujo$ después de ejecutar las $|dirflmin|$ iteraciones, se procede a revisar a través del Algoritmo 4.6, si existen segundos nodos

mezclados en $comtotal$ de las distintas iteraciones que generen $minflujo$ caminos con una menor sumatoria de longitudes respecto a la hallada y almacenada en $minlong$. El arreglo $comtotal$ contiene todos los segundos nodos comunes encontrados en la revisión ejecutada en el Algoritmo 4.5. El Algoritmo 4.6 utiliza a $comtotal$ para intentar mejorar la mínima longitud almacenada en $minlong$. Este algoritmo almacena en el arreglo $longcom$, la $longitud$ de cada uno de los caminos asociados a los segundos nodos dentro de $comtotal$. Si el tamaño de $longcom$ supera o es igual al $minflujo$, se ordena ascendentemente. Si la suma de sus primeras $minflujo$ longitudes es menor que $minlong$, esta suma será el nuevo valor de $minlong$ y los segundos nodos en $comtotal$, asociados a las primeras $minflujo$ longitudes en $longcom$, constituirán el nuevo $mincomun$.

Algoritmo 4.6: Búsqueda_Sobre_Nodos_Comtotal

Entrada: Arreglos $mincomun, comtotal, segnod, longitud$ Escalar $i, minflujo, minlong$
Salida: Arreglo $mincomun$

```
1  Si |mincomun| == minflujo
2      longcom = ∅;
3      longcom(j) = longitud(ic), tal que segnod(i).lista(ic) == comtotal(j),
                                  ∀j = 1, … , |comtotal|;
4      Si |longcom| ≥ minflujo
5          [longcom, poscom] =sort(longcom); // Arreglo poscom guarda posición de
                                             // cada longitud antes de ser ordenado.
6          nuelong = ∑_{m=1}^{minflujo} longcom(m);
7          Si nuelong < minlong
8              minlong = nuelong;
9              mincomun(k) = comtotal(poscom(k)), ∀k = 1, … , minflujo;
10         Fin Si
11     Fin Si
12 Fin Si
13 Retornar mincomun;
```

4.2.3.3 Actualizaciones

El Algoritmo 4.7 calcula el número de caminos válidos hacia un nodo sumidero t. Hasta el momento, en los dos algoritmos anteriores, se ha calculado el conjunto de segundos nodos que identifican los posibles caminos válidos que convergen en el nodo t. Este valor se guarda en la variable $numflujo$ y se obtiene a partir de la longitud del arreglo $mincomun$. El algoritmo inicializa un arreglo de segundos nodos $usados$ en la obtención de los caminos válidos para el sumidero i, con los nodos almacenados en $mincomun$. Los segundos nodos de caminos hacia el sumidero i ya revisados, aunque no todos usados, están en $comtotal$. El algoritmo calcula los segundos nodos cuyos caminos hacia i faltan por revisar y la longitud de cada uno de estos caminos. Estos valores son almacenados en los arreglos $segnodfalta$ y $longnodfalta$, respectivamente. El número de caminos faltantes y que se crearán para el sumidero i es $minflujo - numflujo$. En cada iteración del ciclo entre las líneas 5-12 del algoritmo, se determinan las posiciones de los segundos nodos en $segnodfalta$ cuyos caminos asociados tengan la menor longitud en $longnodfalta$. De estos caminos, se selecciona el primero, del cual se

almacena su segundo nodo en el arreglo $usados$, y se elimina de $segnodfalta$. También se elimina la longitud del camino asociado a este segundo nodo en el arreglo $longnodfalta$.

La última parte del algoritmo calcula los segundos nodos no usados para el sumidero i, a partir de la diferencia entre los arreglos de los segundos nodos en $segnod(i).lista$ y los $usados$. Luego se eliminan los segundos nodos no usados de $segnod(i).lista$ y las rutas no usadas de $rutasdest(i).rutamax$, resultando que cada flujo hacia el sumidero i será el $minflujo$. Además, se adiciona el sumidero i al arreglo $dirflmin$.

Algoritmo 4.7: Actualizaciones

Entrada: Arreglos $rutasdest, segnod, dirflmin, mincomun, comtotal, longitud$
Escalar $i, minflujo$
Salida: Escalar $segnod, rutasdest, dirflmin$

$numflujo = |mincomun|;$
$usados = mincomun;$
$segnodfalta = segnod(i).lista - comtotal;$
$longnodfalta(j) = longitud(k)$, tal que $segnod(i).lista(k) == segnodfalta(j)$, $\forall j = 1, \dots, |segnodfalta|;$
Para $j = 1: minflujo - numflujo$
  $menor = \min(longnodfalta);$
  Adicionar k en arreglo rr, tal que $longnodfalta(k) == menor$, $\forall k = 1, \dots, |longnodfalta|;$
  $seg = segnodfalta(rr(1));$ *//Se toma el primer segundo nodo cuya //posición está en rr porque todas //corresponden a caminos de igual longitud.*
  Adicionar seg a $usados;$
  Eliminar el nodo $segnodfalta(rr(1));$
  Eliminar la longitud $longnodfalta(rr(1));$
Fin Para
$nousados = segnod(i).lista - usados;$
Para $j = 1: |nousados|$
  Sea rr tal que $segnod(i).lista(rr) == nousados(j);$
  Eliminar el segundo nodo $segnod(i).lista(rr);$
  Eliminar el camino $rutasdest(i).rutamax(rr);$
Fin Para
Adicionar i a $dirflmin;$
Retornar $segnod, rutasdest, dirflmin;$

Ejemplo 4.18: Tomando el grafo de la red de comunicaciones de la Figura 4.4, se obtienen los siguientes valores antes de ejecutar el Algoritmo 4.4:

$$flumaxrut = [3,2,2,3,2,4]$$
$$dirflrut = [1,4,6]$$
$$dirflmin = [2,3,5]$$
$$minflujo = 2$$

La Tabla 4.27, presenta el estado de los caminos y la lista de segundos nodos asociados a cada camino y la Figura 4.6 presenta los mismos resultados en los grafos unicast derivados de la ejecución del algoritmo de caminos disyuntos. Se observa como los grafos en (a), (d) y (f), superan el mínimo flujo máximo de 2.

Tabla 4.27 Rutas y listas de segundos nodos para G de 18 nodos y 6 sumideros

i	t_i	$segnod(i).lista$	$rutasdest$		
				$rutamax$	
1	10	[3,4,5]	j	$numruta$	$camino$
			1	1	[1 3 9 10]
			2	2	[1 4 10]
			3	4	[1 5 10]
				$rutamax$	
2	14	[4,6]	j	$numruta$	$camino$
			1	12	[1 4 13 14]
			2	14	[1 6 7 14]
				$rutamax$	
3	15	[2,4]	j	$numruta$	$camino$
			1	3	[1 2 11 15]
			2	7	[1 4 13 15]
				$rutamax$	
4	16	[2,3,4]	j	$numruta$	$camino$
			1	3	[1 2 11 16]
			2	6	[1 3 16]
			3	7	[1 4 8 16]
				$rutamax$	
5	17	[2,3]	j	$numruta$	$camino$
			1	1	[1 2 11 17]
			2	14	[1 3 9 17]
				$rutamax$	
6	18	[2,3,4,5]	j	$numruta$	$camino$
			1	3	[1 2 11 18]
			2	6	[1 3 9 18]
			3	7	[1 4 8 18]
			4	10	[1 5 18]

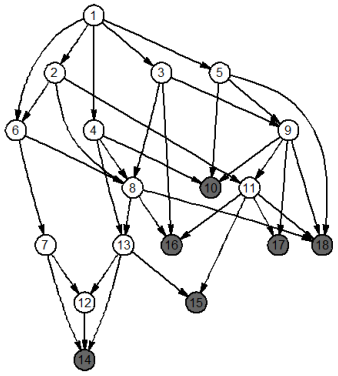


Figura 4.4 Grafo de comunicaciones para 18 nodos y 6 sumideros

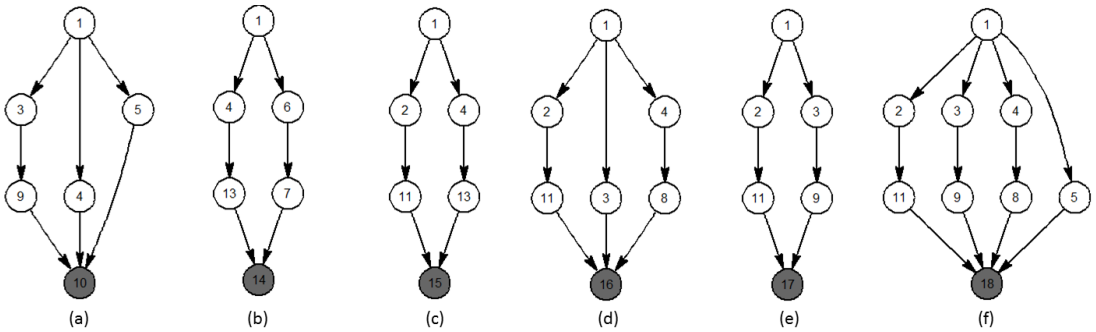


Figura 4.5 Grafos unicast de flujo máximo derivados de G de 18 nodos

Ejemplo 4.19: Con **c**ada iteración del Algoritmo 4.4, sobre los valores presentados en la Tabla 4.27, se generan los resultados presentados en la Tabla 4.28:

Tabla 4.28 Resultados de la Ejecución del Algoritmo 4.4 en G de 18 nodos

i	Algoritmo	Variables de Salida	Valores
1	4.5	$mincomun$	[4]
		$comtotal$	[3,4]
		$longitud$	[4,3,3]
		$minlong$	3
	4.6	$mincomun$	[4]
	4.7	$segnod(i).lista$	[4,5]
		$rutasdest(i).rutamax$	j =2,3
		$dirflmin$	[2,3,5,1]
4	4.5	$mincomun$	[2,3]
		$comtotal$	[2,3,4]
		$longitud$	[4,3,4]
		$minlong$	7
	4.6	$mincomun$	[2,3]
	4.7	$segnod(i).lista$	[2,3]
		$rutasdest(i).rutamax$	j =1,2
		$dirflmin$	[2,3,5,1,4]
6	4.5	$mincomun$	[4,5]
		$comtotal$	[2,3,4,5]
		$longitud$	[4,4,4,3]
		$minlong$	7
	4.6	$mincomun$	[4,5]
	4.7	$segnod(i).lista$	[4,5]
		$rutasdest(i).rutamax$	j =3,4
		$dirflmin$	[2,3,5,1,4,6]

En la Figura 4.6 se muestran los grafos individuales de mínimo flujo máximo para cada sumidero del grado G de 18 nodos. Se observan las rutas de los sumideros incluidos en $dirflrut$.

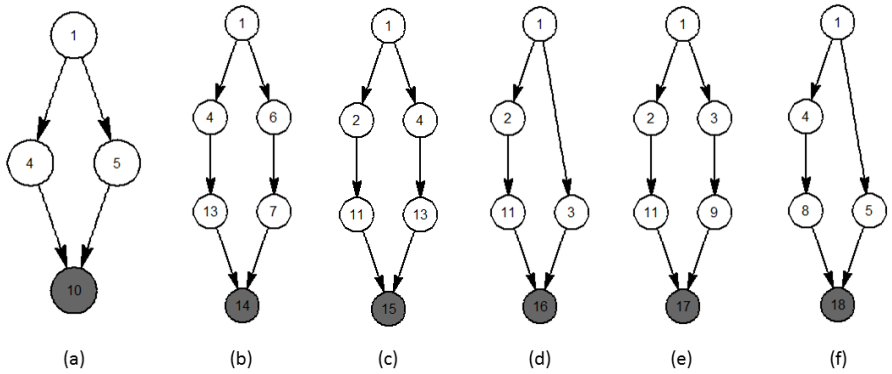


Figura 4.6 Grafos de mínimo flujo máximo para G de 18 nodos

4.2.4 Compilación de rutas en grafo de mínimo flujo máximo

En el Algoritmo 4.8, el arreglo matricial $minmaxflujo$ de tamaño $nn \times nn$ almacena los grafos que corresponden al mínimo flujo para cada sumidero. Es decir, $minmaxflujo$ contiene la mezcla de todos los grafos de las redes de comunicaciones que se forman individualmente a partir de los flujos máximos hallados por cada nodo sumidero. Inicialmente esta matriz está llena de ceros en cada una de sus celdas. En la matriz $minmaxflujo$ se almacenan los caminos válidos desde el nodo fuente s hasta cada sumidero t_i. En este punto, en cada sumidero t_i, confluyen $minflujo$ caminos.

Algoritmo 4.8: Compilación_Rutas_Minmaxflujo

Entrada: Arreglos $rutasdest, dirflmin$ Escalar $nn, minflujo$
Salida: Arreglos $minmaxflujo$

```
1  minmaxflujo(i,j) = 0, (∀i = 1, … , nn, (∀j = 1, … , nn));
2  Para cada sumidero i en dirflmin
3     Para cada camino rutasdest(i).rutamax(j), ∀j = 1, … , minflujo
4        Para cada enlace (k1, k2) ∈ rutasdest(i).rutamax(j).camino
5           minmaxflujo(k1, k2) = 1;
6        Fin Para
7     Fin Para
8  Fin Para
9  Retornar minmaxflujo;
```

4.2.5 Cálculo de nodos de codificación

El Algoritmo 3.10 explicado previamente calcula los nodos de codificación.

Ejemplo 4.20: En la Figura 4.7 se observa el grafo multicast unisesión G' para el grafo G de comunicaciones de 18 nodos. Este grafo resulta de la compilación en la matriz $minmaxflujo$ de los grafos de la Figura 4.6. No se observan nodos de codificación en el grafo G'.

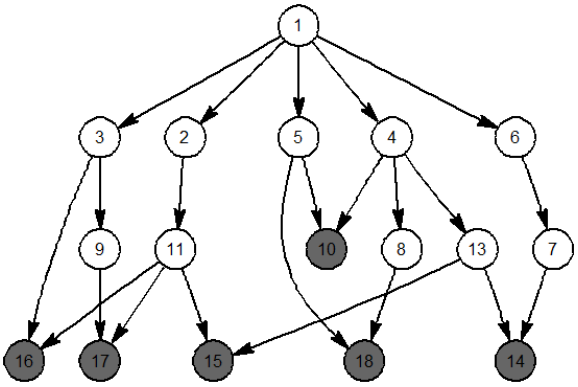


Figura 4.7 Grafo G' multicast unisesión con rutas compiladas para G de 18 nodos

4.3 Reducción del Grafo de Mínimo Flujo Máximo

Dada la situación de la Figura 4.8 (G' obtenido del grafo G de 27 nodos presentado en Figura 4.1), se observa que al nodo sumidero 26 ingresan dos flujos de datos iguales por los enlaces (14,26) y (15,26). Estos corresponden al paquete combinado A+B+C. Esta situación se presenta porque a los nodos 14 y 15, que son codificadores, llegan paquetes que provienen de los segundos nodos 2, 3 y 5, saltando a través de los nodos codificadores 8 y 7, y el nodo de enrutamiento simple 10.

El método a presentar consiste en eliminar algún o algunos enlaces que deriven en la formación de nodos codificadores innecesarios que impidan, con la constitución de los paquetes combinados, una correcta decodificación de los paquetes simples dentro de los nodos sumideros. A continuación se presenta un conjunto de algoritmos que permiten cumplir con este objetivo para muchos grafos multicast de mínimo flujo máximo, sin que se logre, aún, la solución al sistema de ecuaciones lineales sobre la base de Network Coding que se configura en los nodos sumideros para todos los grafos de comunicaciones existentes.

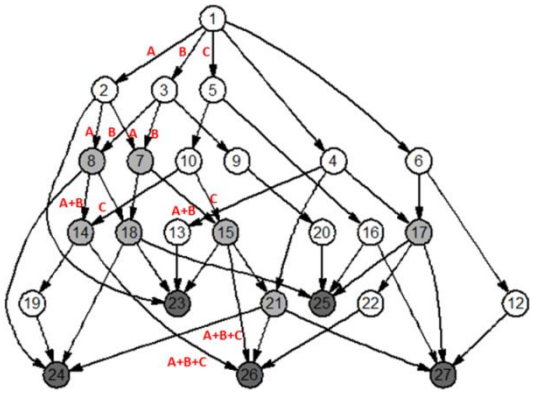


Figura 4.8 Multicast con anulación de entradas en sumidero

4.3.1 Algoritmo de recorte de flujo: opt_minflujo

4.3.1.1 Conceptos previos y principios básicos

Esta función recibe los siguientes datos de entrada:

$minflujo$: Es el valor del mínimo flujo máximo común entre todos los grafos unicast formados después de la reducción inicial y que se conserva después de llevar a cabo la mezcla de estos grafos de flujo máximo.

$nodos\_cod$: Arreglo constituido por los nodos de codificación hallados en el grafo G' resultante de la mezcla de los grafos unicast de flujo máximo.

$minmaxflujo$: Grafo multicast G', resultante de la mezcla de los grafos unicast de flujo máximo. Cada nodo sumidero en este grafo tiene un flujo máximo igual al $minflujo$.

$dest$: Arreglo constituido por los nodos sumideros del grafo multicast G'. Son los mismos nodos sumideros del grafo general de comunicaciones G.

$swseguir$: Bandera booleana que contiene el valor verdadero, lo cual indica que el grafo multicast G' resultante está disponible para seguir siendo revisado y corregido de acuerdo con el criterio de este algoritmo; si el algoritmo cambia la bandera a falso, quiere decir que el grafo G' ya no es susceptible de más revisiones.

La función genera los siguientes datos de salida:

$minmaxflujo$: Aproximación de grafo multicast G' revisado y modificado sin los enlaces que pudieran obstruir la solución para el sistema lineal de ecuaciones basado en Network Coding. Es posible que el sistema arroje que el grafo G' no contiene enlaces que obstruyan la solución y, por ende, la respuesta retornada es igual al grafo entrante. El grafo G' resultante, aún puede no ser el adecuado para una red multicast que conlleve la solución de un sistema lineal de ecuaciones que se resuelva sobre la base de Network Coding.

$nodos\_cod$: Arreglo constituido por los nodos de codificación del grafo G'. Este arreglo puede sufrir cambios si el grafo G' fue modificado en el algoritmo o permanecer idéntico al arreglo entrante al algoritmo, aunque G' sufra cambios. Los nodos en este arreglo se caracterizan por tener más de un enlace entrante. Al presentarse cambios en el grafo G', es posible que solo quede un enlace entrante y, por ende, el nodo deja de ser de codificación para ser un nodo de enrutamiento por almacenamiento y reenvío, exclusivamente.

$swseguir$: Devuelve verdadero si el grafo G' sufre modificaciones; sino, devuelve falso ya que G' no es susceptible de más revisiones y modificaciones.

Las siguientes definiciones serán importantes para el planteamiento de la solución de reducción del grafo multicast de mínimo flujo máximo G'.

Definición 4.1: Sea $a = [a_1, a_2, \dots, a_n]$ una lista de nodos, y a_k un nodo que puede aparecer cero o más veces repetido en a. La expresión $|a|_{=a_k}$ corresponde con el número de apariciones de a_k en a, y la expresión $|a|_{\neq a_k}$ corresponde con el número de elementos en a distintos de a_k.

Definición 4.2: Un penúltimo nodo p de un camino c desde el nodo fuente s hasta el nodo sumidero $t_i \in T$ en un grafo multicast de mínimo flujo máximo, corresponde a un nodo desde el cual emerge el último enlace de c antes de alcanzar t_i. Es decir, es el nodo predecesor de t_i, constituyendo el enlace (p, t_i) de c.

Definición 4.3: Un segundo nodo seg de un camino c desde el nodo fuente s hasta el nodo sumidero $t_i \in T$ en un grafo multicast de flujo máximo, corresponde a un nodo en el cual finaliza el primer enlace de c a partir de s. Es decir, es el nodo sucesor de s, constituyendo el enlace (s, seg) de c.

Definición 4.4: En un grafo multicast de mínimo flujo máximo $minflujo$, una lista de segundos nodos asociados a un penúltimo nodo p ($listseg(p)$) de un nodo sumidero t_i, corresponde con los segundos nodos que forman parte de los caminos que conducen a t_i a través de p. En una $listseg(p)$ pueden existir segundos nodos repetidos, porque se guarda un segundo nodo por cada camino que llega a p, independientemente si ya está en la lista.

Definición 4.5: Sea $a = [a_1, a_2, \dots, a_n]$ una lista de nodos, y a_k un nodo que puede aparecer cero o más veces repetido en a. La lista de a reducida, $reducida(a)$, se obtiene al dejar el nodo a_k de $a, |a|_{=a_k} \bmod 2$ veces en $reducida(a)$. Es decir, si

$$|a|_{=a_k} \bmod 2 = \begin{cases} 0, \text{no habrá apariciones de } a_k \text{ en } reducida(a) \\ 1, \text{se deja una aparición de } a_k \text{ en } reducida(a) \end{cases} \quad (4.23)$$

Definición 4.6: Las funciones $maxnodos$ y $minnodos$, respectivamente, generan como salida el arreglo con más nodos y con menos nodos de entre varios arreglos. Es decir, si $a_1, a_2, \dots, a_n$ son arreglos de cualquier longitud, se definen estas funciones según las expresiones (4.24) y (4.25):

$$minnodos(a_1, a_2, \dots, a_n) = \{a_i | \, |a_i| \leq |a_j|, \forall \, j = 1, \dots, n, j \neq i\} \quad (4.24)$$

$$maxnodos(a_1, a_2, \dots, a_n) = \{a_i | \, |a_i| \geq |a_j|, \forall \, j = 1, \dots, n, j \neq i\} \quad (4.25)$$

Los siguientes lemas serán importantes para explicar el procedimiento aplicado en la reducción del grafo multicast de flujo máximo G'.

Lema 4.1: El número de penúltimos nodos predecesores de un nodo sumidero t_i en un grafo unicast G_i' de flujo máximo, es igual a $minflujo$.

Demostración:
El grafo unicast de flujo máximo obtenido por el procedimiento explicado en la sección 4.2.2, contiene $minflujo$ caminos disyuntos desde s a t_i, denotados por $c_1 = [s, n_{11}, n_{12}, \dots, p_1, t_i], c_2 = [s, n_{21}, n_{22}, \dots, p_2, t_i], \dots, c_{minflujo} = [s, n_{minflujo1}, n_{minflujo2}, \dots, p_{minflujo}, t_i]$, donde n_{qr} denota al r-ésimo nodo del camino c_q y, p_q representa el penúltimo nodo (predecesor de t_i) del camino c_q. Se cumple que $c_1 \cap c_2 \cap \dots \cap c_{minflujo} = \emptyset$, por tanto, $p_1 \neq p_2 \neq \dots \neq p_{minflujo}$, y como conclusión, existirán $minflujo$ penúltimos nodos distintos predecesores de t_i. ■

Lema 4.2: El número de penúltimos nodos predecesores de un nodo sumidero t_i en un grafo multicast G' de mínimo flujo máximo, es igual a $minflujo$.

Demostración:
Sea $t_i \in T$ y sea G_i', el grafo unicast de flujo máximo reducido con valor $minflujo$, desde el nodo fuente s hasta el nodo sumidero t_i, derivado del grafo general de comunicaciones G, el procedimiento explicado en 4.2.2, asegura que en G_i' existen $minflujo$ caminos disyuntos, por

ende, también hay $minflujo$ nodos predecesores del nodo sumidero.

Sean G_i' y G_j' dos grafos unicast individuales reducidos de flujo máximo común $minflujo$ con penúltimos nodos comunes $p_1, p_2, \dots, p_c$, tal que $c \leq minflujo$. Estos son los penúltimos nodos comunes de G_i' y G_j', que contribuyen en forma común a los flujos de t_i y t_j, luego existen $minflujo - c$ penúltimos nodos distintos para cada grafo que contribuyen independientemente al flujo de t_i y t_j. Estos flujos comunes pueden ser paquetes simples o combinaciones de los mismos. Dado los grafos unicast G_i' y G_j' con nodo fuente s y nodos sumideros t_i y t_j, respectivamente, con c penúltimos nodos comunes entre ellos; de los nodos $p_1, p_2, \dots, p_c$ emergen los enlaces $(p_1, t_i), (p_1, t_j), (p_2, t_i), (p_2, t_j), \dots, (p_c, t_i), (p_c, t_j)$ resultantes de la mezcla de los dos grafos durante la conformación del grafo G' multicast.

La mezcla de los dos grafos para la formación del grafo G', no implica la adición de nuevos penúltimos nodos para t_i y t_j. Los $minflujo - c$ penúltimos nodos de t_i y los de t_j, siguen precediendo de forma exclusiva a estos sumideros, además de los c penúltimos nodos comunes. ■

Lema 4.3: Sea G' un grafo multicast de mínimo flujo máximo común $minflujo$, y sea $\mathcal{C}(G_i')$ el conjunto de caminos desde s hasta $t_i \in T$ en G', luego si $|\mathcal{C}(G_i')| = minflujo$, todos los caminos desde s hasta t_i son disyuntos.

Demostración:
Sean G_i' y G_j' grafos unicast de flujo máximo $minflujo$ desde s hasta t_i y t_j, con $t_i, t_j \in T$, respectivamente; por definición de flujo máximo basado en el procedimiento en la sección 4.2.2, el número de caminos disyuntos desde s hasta t_i y t_j es el $minflujo$.

Sea $\mathcal{C}(G_i')$ y $\mathcal{C}(G_j')$ el conjunto de caminos disyuntos de G_i' y G_j', respectivamente, y asumiendo que $\left|\mathcal{C}(G_i') \cup \mathcal{C}(G_j')\right| = 2 * minflujo$ número de caminos resultantes en G' despúes de mezclar a G_i' y G_j' en G'. Luego, estos caminos no se entrelazan, y $\mathcal{C}(G_i') \cap \mathcal{C}(G_j') = \emptyset$ en G', por tanto $|\mathcal{C}(G_i')| = minflujo$. Sea $c_q(G_i')$ el q-ésimo camino del grafo unicast G_i' dentro de G', por el procedimiento explicado en la sección 4.2.2, se infiere que:

$$\bigcap_{q=1}^{minflujo} c_q(G_i') = \emptyset \tag{4.26}$$

Es decir, todos los caminos de G_i' dentro de G', desde s hasta t_i son disyuntos. ■

Después de obtener el flujo máximo común (mínimo flujo máximo, $minflujo$) para todos los grafos que permiten la comunicación unicast entre el nodo fuente s y cada sumidero t_i en G, se asume que todos tienen un $minflujo$ de caminos disyuntos que conducen a lograr el flujo máximo común en cada sumidero. Al llevar a cabo la reducción de los grafos unicast, a partir del grafo general de comunicaciones G, para que en el nodo sumidero converjan $minflujo$ caminos disyuntos, el número de nodos predecesores (penúltimo nodo en cada camino) para cada t_i, debe ser el valor del $minflujo$, de acuerdo con el Lema 4.1. Según el Lema 4.2, al mezclar los grafos

unicast de flujo máximo común para obtener el grafo de comunicación multicast G', los penúltimos nodos permanecen siendo iguales; no obstante, se pueden generar nuevos caminos entre el nodo fuente y cada nodo sumidero al configurarse los nodos de codificación, resultado del ingreso de dos o más enlaces al mismo nodo intermedio, consecuencia de la adhesión de las rutas de grafos unicast que pasan por este nodo. Esto significa que se conformarían nuevos caminos y, no necesariamente disyuntos, con la posibilidad que al llevar los paquetes al nodo sumidero, se anularían; no permitiendo la solución del sistema y, por ende, la no entrega de todos los paquetes simples originales enviados desde el nodo fuente.

Lema 4.4: El número total de elementos (nodos) pertenecientes a todas las $listseg$ asociadas a los penúltimos nodos de un sumidero t_i en un grafo multicast G' de mínimo flujo máximo $minflujo$, es igual al número total de caminos desde s hasta t_i.

Demostración:
Sea G' un grafo multicast de mínimo flujo máximo configurado después de mezclar los $|T|$ grafos unicast de flujo máximo, y sea N el número total de caminos desde s hasta $t_i \in T$. Por el Lema 4.2, el número de penúltimos nodos para cualquier t_i es $minflujo$ y, $minflujo \leq N$.

Si $minflujo = N$, por el Lema 4.3, todos los caminos desde s hasta t_i son disyuntos y como consecuencia, todos están configurados con nodos únicos, lo cual quiere decir que el número total de segundos nodos en las $listseg$ asociadas a los penúltimos nodos de t_i es N.

Si $minflujo < N$, los caminos desde s hasta t_i pueden tener nodos comunes (no disyuntos). Sea p_i^j, el j-ésimo penúltimo nodo predecesor de t_i, tal que en este convergen r_j-caminos ($1 \leq r_j \leq N$). Estos r_j-caminos que convergen en p_i^j pueden tener segundos nodos distintos o iguales en sus trayectorias, y de igual forma, los paquetes que están en capacidad de transportar, pueden ser simples, combinados o vacíos, como consecuencia de haber cruzado por nodos codificadores que realizan previamente operaciones de combinaciones lineales. Sea seg_m, el segundo nodo del camino m ($1 \leq m \leq r_j$) que alcanza a t_i, se puede definir una lista de segundos nodos asociada con cada penúltimo nodo p_i^j, tal como:

$$listseg\left(p_i^j\right) = \{seg_m | 1 \leq m \leq r_j\} \tag{4.27}$$

En este conjunto pueden existir segundos nodos repetidos $seg_{m_1} = seg_{m_2}$, donde $m_1 \neq m_2$, ya que los caminos pueden tener nodos comunes por no ser disyuntos.

De (4.27), se obtiene que $\left|listseg\left(p_i^j\right)\right| = r_j$; es decir, el número de segundos nodos en la $listseg$ del penúltimo nodo p_i^j, es el mismo número de caminos que convergen en p_i^j. Por el Lema 4.2, existen $minflujo$ p_i^j; o lo que es lo mismo, hay $minflujo$ conjuntos de r_j-caminos que parten desde s e ingresan a t_i, tomando de cada uno el segundo nodo $seg_m, 1 \leq m \leq r_j$, para configurar $listseg\left(p_i^j\right)$. Luego el número de caminos que viajan desde s hasta t_i es:

$$\sum_{j=1}^{minflujo} r_j = N = \sum_{j=1}^{minflujo} \left|listseg\left(p_i^j\right)\right| \tag{4.28}$$

La Figura 4.9 muestra los $minflujo$ conjuntos de r_j-caminos, donde cada conjunto converge en cada j-penúltimo nodo p_i^j para un total de N-caminos desde s hasta t_i. ■

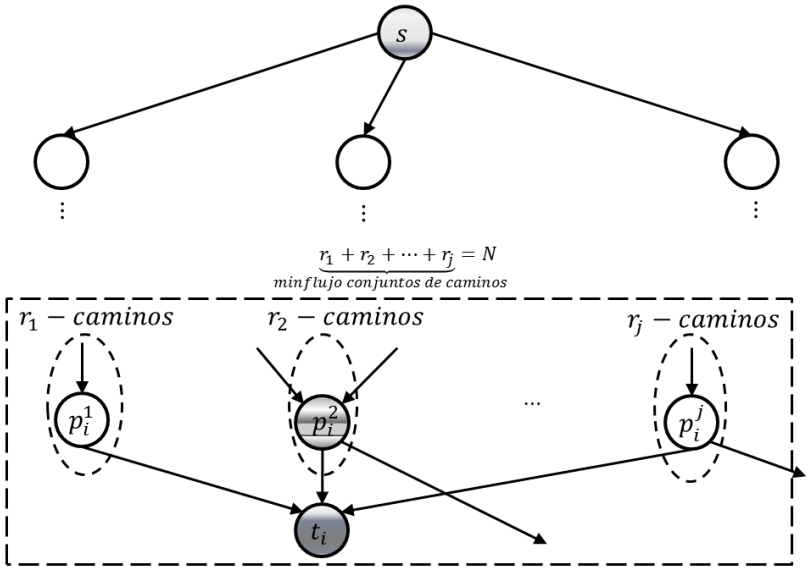


Figura 4.9 Ilustración del Lema 4.4

Lema 4.5: Sean p_1 y p_2 dos penúltimos nodos que preceden al nodo sumidero t_i. Sean $listseg(p_1)$ y $listseg(p_2)$ las listas de segundos nodos de p_1 y p_2, respectivamente. Si $reducida(listseg(p_1)) = reducida(listseg(p_2))$, los flujos de paquetes que ingresan por p_1 y p_2 se anulan.

Demostración:
Sea $reducida\left(listseg(p_1)\right) = reducida\left(listseg(p_2)\right) = [seg_1, seg_2, \dots, seg_m]$, y sean A_j las etiquetas de los paquetes que emergen a través de los enlaces $(s, seg_j), 1 \leq j \leq m$. Los paquetes salientes de p_1 y p_2, y resultantes de las combinaciones lineales $\sum_{j=1}^{m} A_j$ que se configuran a lo largo de los trayectos hacia p_1 y p_2, serán las mismas para ambos penúltimos nodos. Ambos paquetes combinados entrarán en el nodo sumidero t_i y se anularán durante la decodificación.

La Figura 4.10 muestra la forma en que egresan los paquetes desde el nodo fuente s, cómo son llevados hasta los penúltimos nodos p_1 y p_2, y entregados en el nodo sumidero t_i. ■

Lema 4.6: El número de enlaces de salida que parten del nodo s y llegan a los segundos nodos en un grafo multicast de mínimo flujo máximo G', no debe ser menor que el $minflujo$.

Demostración:
Partiendo del supuesto que por cada enlace solo se puede transportar una unidad de datos (un

paquete) y, que además, el mínimo flujo máximo común es $minflujo$, se infiere que a cada nodo t_i, le llegan $minflujo$ paquetes distintos simultáneamente que se originan en s. Es decir, que se necesitan, por lo menos, $minflujo$ enlaces de salida desde s hasta los segundos nodos que transporten un único paquete a la vez. Estos paquetes pueden ser simples o combinados, pero todos de tamaño una unidad de datos. ■

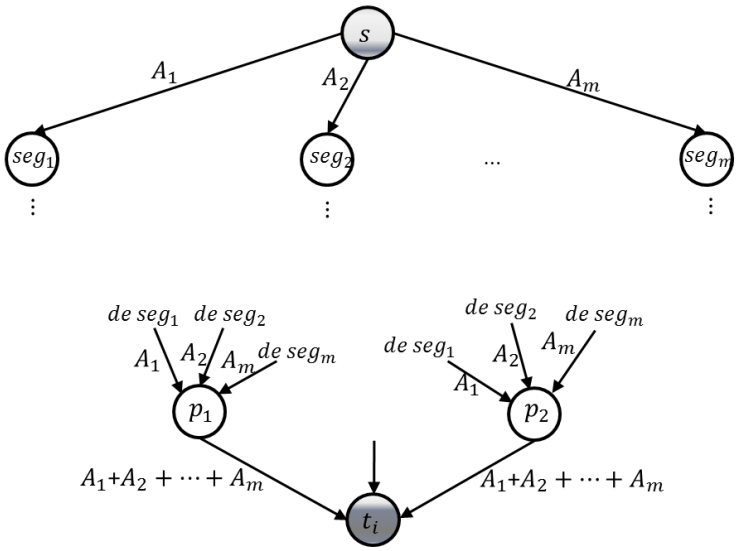


Figura 4.10 Ilustración del

Lema **4.5**

Ejemplo 4.21: En la Figura 4.11, se observan, en (a) y (b), los grafos unicast de flujo máximo con $minflujo = 4$.

En (a), para alcanzar el sumidero 23, se encuentran los caminos:
$c_1 = [1,2,23], c_2 = [1,3,\mathbf{8},\mathbf{18},23], c_3 = [1,4,13,23], c_4 = [1,5,10,15,23]$.

En (b), para alcanzar el sumidero 24, se encuentran los caminos:
$c_1 = [1,2,\mathbf{8},24], c_2 = [1,3,7,\mathbf{18},24], c_3 = [1,4,21,24], c_4 = [1,5,10,14,19,24]$.

Los cuatro caminos de cada grafo unicast son disyuntos; es decir, no existen nodos repetidos en ningún camino para el mismo grafo.

En cambio, cuando se mezclan estos dos grafos, con el fin de ir configurando el grafo multicast, se observan nuevos caminos que surgen por la existencia de nodos comunes y enlaces comunes entre caminos distintos. Se observa que los nodos 8 y 18 están repetidos en caminos de los dos grafos unicast y esto conlleva la creación de nuevos caminos que conducen a los nodos sumideros 23 y 24. Estos nodos se convierten en nodos de codificación en el momento de la mezcla.

Los caminos adicionales que en (c) se forman para alcanzar el sumidero 23, son:
$c_5 = [1,2,8,18,23], c_6 = [1,3,7,18,23]$.

Los caminos adicionales que en (c) se forman para alcanzar el sumidero 24, son:
$c_5 = [1,3,8,18,24], c_6 = [1,2,8,18,24]$.

También se observa que los penúltimos nodos en los grafos unicast que conducen a los nodos sumideros 23 y 24, se mantienen para el grafo multicast resultante de la mezcla.

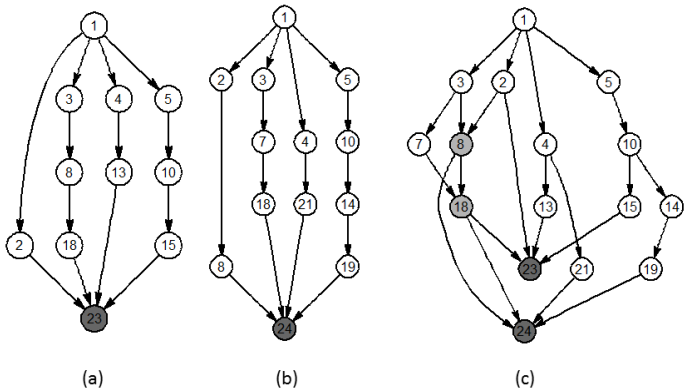


Figura 4.11 (a) y (b) Grafos unicast de flujo máximo 4. (c) Grafo multicast resultante de la mezcla de (a) y (b) de flujo máximo común 4

El método de Optimización de Flujo se presenta en el Algoritmo 4.9 y está constituido de las tres funciones, que a continuación se describen:

Algoritmo 4.9: Optimización_Flujo

Entrada: Arreglos $minmaxflujo, nodos\_cod, dest, segnodtot$ Escalar $minflujo, swseguir$
Salida: Arreglo $minmaxflujo, nodos\_cod$ Escalar $swseguir, swp$

```
1  [tdrmin, contarseg]=Derivación_Todas_Rutas_Desde_s_Hasta_t_i(
                                   minmaxflujo, dest, segnodtot, minflujo);
2  codseg=Construcción_Arreglo_codseg(tdrmin, nodos_cod);
3  [minmaxflujo, nodos_cod, swseguir, swp]=
   Eliminación_Enlace_Evasión_Convergencia(
   minmaxflujo, codseg, tdrmin, nodos_cod, segnodtot, contarseg, swseguir);
4  Retornar minmaxflujo, nodos_cod, swseguir, swp;
```

4.3.1.2 Derivación de todas las rutas posibles desde el nodo fuente hasta el nodo sumidero

Se construye la estructura $tdrmin(i)$ para cada nodo sumidero t_i, cuyo número de caminos desde el nodo s hasta t_i sea mayor que el $minflujo$. Si el número de rutas para alcanzar un sumidero t_i es el mismo $minflujo$ (Lema 4.3), este sumidero no se debe revisar, ya que todas serán disyuntas y es lo que se exige; lo contrario indicaría que algunos caminos obtenidos tienen nodos comunes y, por ende, se deben corregir.

Formalmente, sea $tdrdest$ todos los caminos posibles entre s y t_i, solo se construye $tdrmin(i)$ para un t_i, si y solo sí, $|tdrdest| > minflujo$. Si $|tdrdest| = minflujo$, ya se cumple el número de caminos disyuntos que satisfacen el mínimo flujo máximo común (según el Lema 4.4). Cada $tdrmin(i)$ tiene la forma:

$tdrmin(i)$
    $tdrmn(j)$
        $camino$
    $extremos(k)$
        pen
        $listseg$

En esta estructura se observan los siguientes componentes:

i: Índice del nodo sumidero t_i en revisión.

j: Índice que señala al j-ésimo camino hacia el nodo sumidero t_i desde el nodo fuente s.

$camino$: Nodos adyacentes a través de enlaces que especifican un camino desde s hasta t_i.

$extremos$: Arreglo que contiene la estructura constituida por los penúltimos nodos precedentes a cada nodo sumidero t_i y, la lista de los segundos nodos o nodos adyacentes al nodo fuente que hacen parte de cada $camino$ que conduce a un penúltimo nodo.

k: Índice, cuyo valor máximo es el $minflujo$, que representa el número de penúltimo nodo (o precedente) de un nodo sumidero y como consecuencia, el índice con el que se referencia cada componente en el arreglo $extremos$. El valor máximo de k está acorde con el Lema 4.1 y el Lema 4.2.

pen: Corresponde al penúltimo nodo predecesor del nodo sumidero t_i en la posición k-ésima del arreglo $extremos$.

$listseg$: Cada k-ésimo penúltimo nodo contiene asociado un conjunto de segundos nodos que se obtienen de todos los caminos (uno o más caminos) que convergen en éste. Se puede configurar una lista tal como:

$$pen(k): seg_1, seg_2, \dots, seg_{|listseg_k|} \tag{4.29}$$

En (4.29), cada seg_m corresponde al segundo nodo que utiliza el m-ésimo $camino$ que llega al nodo $pen(k)$. Pueden existir un seg_{m_1} y un seg_{m_2}, tal que $seg_{m_1} = seg_{m_2}$, donde m_1 y m_2 son dos caminos distintos que conducen a $pen(k)$, pero no disyuntos.

Paralelamente a esta estructura vectorial, se define de forma temporal e independiente, la estructura $extremos(k)$ con la misma forma de su análoga en $tdrmin(i)$.

Según el Lema 4.4, la relación en (4.30) es válida:

$$\sum_{k=1}^{minflujo} |listseg_i(k)| = |tdrmin(t_i).tdrmn| \tag{4.30}$$

Se observa que, después de haber mezclado los grafos unicast de mínimo flujo máximo, la sumatoria del total de segundos nodos en las listas $listseg$ pertenecientes a los penúltimos nodos asociados a cada sumidero t_i, es igual al número de rutas desde s hasta t_i.

Para asegurar lo propuesto en el Lema 4.6, y que el procedimiento planteado no elimine enlaces iniciales que conduzcan a que queden menos enlaces que el valor del $minflujo$, se mantiene la matriz $contarseg$ de dimensión $|tdrmin| \times |segnodtot|$. Esta matriz almacena el número de veces que cada segundo nodo es utilizado dentro de los r_j-caminos que convergen en el penúltimo nodo p_i^j precedente del sumidero t_i. Si $contarseg(i, seg) = n$, quiere decir que el segundo nodo seg aparece en n-caminos del conjunto de r_j-caminos que llegan a p_i^j previo al sumidero t_i. Esto se lleva a cabo revisando el conjunto de caminos que llegan a t_i almacenados en el $tdrdest$ correspondiente.

El Algoritmo 4.10 resume los pasos para el cálculo de la estructura $tdrmin$ compuesta por los caminos, penúltimos nodos y las listas de segundos nodos asociadas a estos. Se ejecutan cíclicamente, de acuerdo con el número de nodos sumideros presentes en el grafo multicast G', los siguientes pasos:

a. Cálculo de todos los caminos posibles desde el nodo s hasta un sumidero t_d en $tdrdest$. Si el número de caminos es superior al $minflujo$, se procede con (b), (c), (d) y (e). Sino, se continúa con el siguiente sumidero t_{d+1}.
b. Guardar $tdrdest$ en $tdrmin$.
c. Calcular la unión de los penúltimos nodos de los caminos en $tdrdest$ para t_d y almacenar, de forma única y ordenados topológicamente, cada penúltimo nodo en el arreglo $penultimo$. De esta forma, quedan solo $minflujo$ penúltimos nodos almacenados por cada sumidero t_d (Lema 4.2). También se calcula la matriz $contarseg$.
d. Almacenamiento del k-ésimo penúltimo ($penultimo(k)$) nodo en $extremos(k).pen$ e inicialización de cada lista $extremos(k).listseg$ como un conjunto vacío.
e. Buscar en cada camino de $tdrdest$, el segundo nodo e insertarlo en la $listseg$ respectiva de acuerdo con el penúltimo nodo de este camino. De esta forma queda actualizado $extremos(k)$ para asignarlo a $tdrmin(i).extremos$.

Algoritmo 4.10: Derivación_Todas_Rutas_Desde_s_Hasta_t_i

Entrada: Arreglos $minmaxflujo, dest, segnodtot$ Escalar $minflujo$
Salida: Arreglo $tdrmin, contarseg$

```
tdrmin = ∅;
i = 1;
s = 1;
Para d = 1:|dest|
    tdrdest = todas_las_rutas(s, dest(d), minmaxflujo);
    Si |tdrdest| > minflujo
        tdrmin(i).tdrmn = tdrdest;
        contseg(seg) = 0, ∀ seg = 1, …, |segnodtot|;
        penultimo = ∅;
```

```
14        Para k = 1: |tdrdest|
15           Agregar penúltimo nodo de tdrdest(k).camino a penultimo si no
                 está;
16           contseg(seg) + +, tal que segnodtot(seg) = tdrdest(k).camino(2);
17        Fin Para
18        contarseg(i) = contseg; //Matriz de |tdrmin| × |segnodtot|
19        Para k = 1: |penultimo|
20           extremos(k).pen = penultimo(k);
21           extremos(k).listseg = ∅;
22        Fin Para
23        Para j = 1: |tdrdest|
24           Buscar un pospen tal que extremos(pospen).pen coincida con el
             penúltimo nodo de tdrdest(j).camino;
25           Adicionar tdrdest(j).camino(2) a extremos(pospen).listseg;
26        Fin Para
27        tdrmin(i).extremos = extremos;
28        i + +;
29     Fin Si
30  Fin Para
31  Retornar tdrmin, contarseg;
```

Ejemplo 4.22: De la Figura 4.8, se extraen todos los *caminos* que se forman desde el nodo fuente 1 hasta cada nodo sumidero después de la mezcla de los grafos unicast de flujo máximo. Estos *caminos* se presentan en la Tabla 4.29. En la Tabla 4.30, se observan los arreglos *extremos* para cada nodo sumidero. Se observa que cada lista de extremos es constituida por el penúltimo nodo *pen* y la lista *listseg* de segundos nodos que están en los caminos que conducen a este penúltimo nodo y que, por ende, llegan al nodo sumidero.

Tabla 4.29 Caminos que llegan a cada sumidero desde el nodo fuente en G' de la Figura 4.8

Caminos de 1 a 23				
1	2	7	15	**23**
1	2	7	18	**23**
1	2	8	18	**23**
1	2	**23**		
1	3	7	15	**23**
1	3	7	18	**23**
1	3	8	18	**23**
1	4	13	**23**	
1	5	10	15	**23**

Caminos de 1 a 24					
1	2	7	15	21	**24**
1	2	7	18	**24**	
1	2	8	14	19	**24**
1	2	8	18	**24**	
1	2	8	**24**		
1	3	7	15	21	**24**
1	3	7	18	**24**	
1	3	8	14	19	**24**
1	3	8	18	**24**	
1	3	8	**24**		
1	4	21	**24**		
1	5	10	14	19	**24**
1	5	10	15	21	**24**

Caminos de 1 a 25				
1	2	7	18	**25**
1	2	8	18	**25**
1	3	7	18	**25**
1	3	8	18	**25**
1	3	9	20	**25**
1	4	17	**25**	
1	5	16	**25**	
1	6	17	**25**	

Caminos de 1 a 26					
1	2	7	15	21	**26**
1	2	7	15	**26**	
1	2	8	14	**26**	
1	3	7	15	21	**26**
1	3	7	15	**26**	
1	3	8	14	**26**	
1	4	17	22	**26**	
1	4	21	**26**		
1	5	10	14	**26**	
1	5	10	15	21	**26**
1	5	10	15	**26**	

Caminos de 1 a 27					
1	2	7	15	21	**27**
1	3	7	15	21	**27**
1	4	17	**27**		
1	4	21	**27**		
1	5	10	15	21	**27**
1	5	16	**27**		
1	6	12	**27**		
1	6	17	**27**		

1	6	17	22	**26**	

Tabla 4.30 $tdrmin(i).extremos$ para cada nodo sumidero

i	t_i	i	t_i	i	t_i	i	t_i	i	t_i
1	**23**	**2**	**24**	**3**	**25**	**4**	**26**	**5**	**27**
pen	*listseg*	*pen*	*listseg*	*pen*	*listseg*	*pen*	*listseg*	*pen*	*listseg*
2	2	8	2 3	16	5	14	2 3 5	12	6
13	4	18	2 2 3 3	17	4 6	15	2 3 5	16	5
15	2 3 5	19	2 3 5	18	2 2 3 3	21	2 3 4 5	17	4 6
18	2 2 3 3	21	2 3 4 5	20	3	22	4 6	21	2 3 4 5

4.3.1.3 Creación de la estructura codificadores-segundos nodos ($codseg$)

La estructura $codseg$ consiste de un arreglo que contiene en cada posición la lista resumida de los nodos codificadores y segundos nodos de los caminos del grafo multicast a reducir. Para calcular $codseg$ se revisan los conjuntos de caminos asociados a cada nodo sumidero almacenados en $tdrmin$. Si llegan a existir listas iguales, estas se almacenan una vez. La forma de la estructura es:

$codseg(i)$
    $nodos$

Cada posición i-ésima de $codseg$ está formada por una lista de nodos codificadores y el segundo nodo del camino de donde se extrajo.

El método para calcular $codseg$ consiste en recorrer el $tdrmin(i)$ asociado a cada nodo sumidero t_i y aplicar los pasos (a) y (b) descritos en el Algoritmo 4.11 y, el paso (c) descrito en el Algoritmo 4.12:

a. Recorrer los $minflujo\ extremos$ asociados a $tdrmin(i)$ para hallar dos $listseg$ reducidas (En la sección 4.3.3 se explica el algoritmo para reducir listas) iguales. Si se hallan, quiere decir que estos dos caminos transportarán los mismos paquetes (simples o combinados) hasta los penúltimos nodos predecesores del nodo sumidero t_i, y que se anularán en este último, por ende, no contribuirán a la decodificación. Al cumplirse la condición de igualdad entre dos $listseg$ reducidas, se aplica el procedimiento explicado en el acápite (c).
b. Si no se hallan dos $listseg$ reducidas iguales, se busca la primera $listseg$ reducida vacía del arreglo $tdrmin(i).extremos$; si se obtiene, se procede con el procedimiento expuesto en el acápite (c). Esta acción es necesaria, dado que si la lista $listseg$ reducida corresponde con un conjunto vacío, el enlace entrante al nodo t_i que conduce esta lista, no aportará ningún paquete a la solución en este nodo sumidero.
c. Se procede a hallar los nodos de codificación que se encuentran en cada camino de $tdrmin$ que conducen al nodo sumidero t_i que cumpla la condición en (a) o (b). Si la lista de nodos de codificación ($codint$) en el j-ésimo camino ($tdr.tdrmn(j).camino$) no está vacía, se anexa el segundo nodo del $camino$ a la lista $codint$, creando el arreglo $nodos$. A continuación se verifica que el arreglo $codseg$ no esté vacío (línea 7) y, por último, se verifica si esta lista de $nodos$ es subconjunto del componente ik de $codseg$ o viceversa. De ser así, se actualiza $codseg(ik).nodos = maxnodos(nodos, codseg(ik).nodos)$, ya que para las verificaciones posteriores se debe tener el mayor espectro de nodos codificadores en $codseg(ik)$. De no presentarse ninguna de las situaciones anteriores, se adiciona $nodos$ al arreglo $codseg$.

De esta forma, queda constituido el arreglo $codseg$ para todos los nodos sumideros t_i en G' que contienen más caminos entrantes que $minflujo$ y que, por tanto, no son disyuntos.

Algoritmo 4.11: Construcción_Arreglo_$codseg$: Cumplimiento de condiciones

Entrada: Arreglos $tdrmin, nodos\_cod$
Salida: Arreglo $codseg$

```
1  codseg = ∅;
2  Para i = 1:|tdrmin|
3     Para cada dos listseg reducidas en tdrmin(i) iguales
4        codseg=Conscodseg(codseg,nodos_cod,tdrmin(i));
5     Fin Para
6     Para cada listseg reducida en tdrmin(i) vacía
7        codseg=Conscodseg(codseg,nodos_cod,tdrmin(i));
8     Fin Para
9  Fin Para
10 Retornar codseg;
```

Algoritmo 4.12: Conscodseg: Inserción_Segundos_Nodos

Entrada: Arreglos $codseg, nodos\_cod, tdr$
Salida: Arreglo $codseg$

```
1  ij = |codseg| + 1;
2  Para j = 1:|tdr.tdrmn| //Recorrido sobre todos los caminos en tdr
3     codint = nodos_cod ∩ tdr.tdrmn(j).camino;
4     Si codint ≠ ∅
5        nodos = codint ‖ tdr.tdrmn(j).camino(2);
6        swc = false;
7        Si ij > 1
8           ik = 1;
9           Mientras ~swc ∧ ik ≤ |codseg|
10             Si nodos ⊆ codseg(ik).nodos ∨ codseg(ik).nodos ⊆ nodos
11                codseg(ik).nodos = maxnodos(nodos, codseg(ik).nodos);
12                swc = true;
13             Sino
14                ik++;
15             Fin Si
16          Fin Mientras
17       Fin Si
18       Si ~swc
19          codseg(ij).nodos = nodos;
20          ij++;
21       Fin Si
22    Fin Si
23 Fin Para
24 Retornar codseg;
```

Ejemplo 4.23: Siguiendo con el Ejemplo 4.22, se toman los índices $i = 1, \dots, 5$ correspondientes a

los sumideros $t_i = 23, \ldots, 27$, y se genera el arreglo $codseg$ listado en la Tabla 4.31.

Tabla 4.31 Arreglo $codseg$ para primera iteración con G' de 27 nodos

cod			*seg*
7	15	21	2
7	18		2
8	18		2
7	15	21	3
7	18		3
8	18		3
15	21		5
8	14		2
8	14		3
21			4
14			5
17			4
17			6

4.3.1.4 Eliminación de enlace para evitar convergencia de caminos al mismo sumidero

El objetivo del procedimiento es eliminar un enlace para evitar que dos caminos distintos converjan al mismo nodo sumidero t_i con la misma $listseg$ reducida. En el Algoritmo 4.13 se resume la ejecución para lograr este objetivo. Se ejecutan los pasos para cada lista de nodos codificadores y segundo nodo en cada posición $h \leq |codseg|$, controlados también por la bandera swc que se mantendrá inactiva (false), mientras no se cumpla la condición para la eliminación del enlace propósito de este procedimiento.

Se construye un arreglo de dos nodos ($conj_2$), formado con el nodo de codificación inicial y el segundo nodo de la lista en cada posición en $codseg$. También se inicializa vacía la lista $sumiseg$ de posiciones de $tdrmin$, correspondientes a los sumideros t_i que cumplirán la condición para eliminación del enlace.

Para cada $codseg$ revisado, se activa la bandera sw y se itera con la variable i sobre el arreglo $tdrmin$ que contiene los conjuntos de los caminos que conducen a t_i. La bandera sw se desactiva, si para un conjunto de caminos pertenecientes a un sumidero t_i, se obtiene un conjunto vacío a partir de la función xor de sus segundos nodos o *xorvect* (sección 4.3.2).

Algoritmo 4.13: Eliminación_Enlace_Evasión_Convergencia

Entrada: Arreglos $minmaxflujo, codseg, tdrmin, nodos\_cod, segnodtot, contarseg$
Escalar $swseguir$
Salida: Arreglos $minmaxflujo, nodos\_cod, swseguir$ Escalar: swp

1 $swc = false;$
2 $h = 1;$
3 **Mientras** $\sim swc \wedge h \leq |codseg|$
4 $sw = true;$
5 $i = 1;$
6 $ultimo = |codseg(h).nodos|;$
7 $conj_2 = [codseg(h).nodos(1), codseg(h).nodos(ultimo)];$ *//Mezcla de los nodos //extremos del* $codseg$ *evaluado*
8 $sumiseg = \emptyset;$

```
9     Mientras ~sw ∧ i ≤ |tdrmin|
10        [swe, peneleg, rutesc, tcod]=Determinación_Ruta_Eliminar(
                                          tdrmin, nodos_cod, codseg, conj_2, i);
11        Si swe
12           [tdrmin, zxor]=Verificación_Ruta_Eliminar(tdrmin, peneleg, i);
13           Si zxor == ∅
14              sw = false;
15           Sino
16              Adicionar i a sumiseg;
17              selsum = i;
18              selrut = rutesc;
19              i++;
20           Fin Si
21        Sino
22           i++;
23        Fin Si
24    Fin Mientras
25    Si i − |tdrmin| == 1
26       contarseg(p, q) = − −, tal que p ∈ sumiseg, y segnodtot(q) == tsec;
27       Si |contarseg(p)|_≠0 < minflujo para algún p ∈ sumiseg
28          contarseg(p, q) = + +, tal que p ∈ sumiseg, y segnodtot(q) == tsec;
29          h++;
30       Sino
31          swc = true;
32       Fin Si
33    Sino
34       h++;
35    Fin Si
36 Fin Mientras
37 swp = false;
38 Si swc
39    [minmaxflujo, nodos_cod]=Actflujo(
                  minmaxflujo, nodos_cod, tdrmin(selsum).tdrmn(selrut).camino, tcod);
40 Sino
41    swseguir = false;
42    Si |segnodtot| == minflujo ∧ tdrmin ≠ ∅
43       [tcod, selsum, selrut, swp]=Verdet(tdrmin, minflujo, dest, segnodtot, nodos_cod);
44       Si swp
45          [minmaxflujo, nodos_cod]=Actflujo(
                  minmaxflujo, nodos_cod, tdrmin(selsum).tdrmn(selrut).camino, tcod);
         Fin Si
46    Fin Si
47 Fin Si
48 Retornar minmaxflujo, nodos_cod, swseguir, swp;
```

Dentro del ciclo, se ejecutan los siguientes dos procedimientos:

a. Determinación de la Ruta a Eliminar: De acuerdo con el Algoritmo 4.14, se ejecuta un ciclo con la variable j (índice de los caminos hacia cada sumidero t_i) sobre el arreglo $tdrmin(i).tdrmn$ para buscar si en sus $caminos$ existen nodos de codificación. Si contiene nodos de codificación, estos se concatenan con el segundo nodo del camino creando el arreglo $nodos$; además de obtener el arreglo $conj_1$ formado por el primer nodo de codificación y el segundo nodo del camino, que se utilizará en conjunto con $conj_2$, para verificar si realmente existe un camino coincidente con $codseg$.

A continuación, se obtienen los arreglos $conj$ y $subconj$ a través de las siguientes expresiones:

$$conj = maxnodos(codseg(h).nodos, nodos) \tag{4.31}$$

$$subconj = minnodos(codseg(h).nodos, nodos) \tag{4.32}$$

Si se logra el cumplimiento de la doble condición $subconj \subseteq conj$ y $conj_1 = conj_2$, se deduce que se encontró un camino j en el $tdrmin(i)$. Es decir, con la primera condición, para el nodo sumidero t_i, se encontró el camino $tdrmin(i).tdrmn(j).camino$ que cumple el emparejamiento en sus nodos de codificación componentes y el segundo nodo con el $codseg(h).nodos$. Además, la segunda condición se usa para completar de asegurar que este es el camino, ya que el primer nodo codificador y el segundo nodo, tanto del $codseg(h)$ y del camino escogido, coinciden uno a uno entre sí. Cumplidas estas condiciones, se almacena la posición del camino j en el arreglo $peneleg$, se guarda el camino escogido (nuevamente j) en la variable $rutesc$, y el primer nodo codificador del arreglo $codseg(h).nodos$ (que es el mismo del camino escogido) en la variable $tcod$ (este va a servir de base, si se mantiene la prueba, para eliminar el enlace que obstruye la solución). Además, se activa la bandera swe para especificar que por lo menos el camino $tdrmn(i).tdrmn(j).camino$ cumplió con la condición.

Algoritmo 4.14: Determinación_Ruta_Eliminar

Entrada: Arreglos $tdrmin, nodos\_cod, codseg, conj_2$ Escalares i
Salida: Arreglo $swe, peneleg, rutesc, tcod$

```
1  swe = false;
2  peneleg = ∅;
3  Para cada camino tdrmin(i).tdrmn(j), j = 1, … , |tdrmin(i).tdrmn|
4      codint = nodos_cod ∩ tdrmin(i).tdrmn(j).camino;
5      Si codint ≠ ∅
6          nodos = [codint, tdrmin(i).tdrmn(j).camino(2)]; //Mezcla de nodos de
           //codificación en camino j y segundo nodo de camino j
7          ultimo = |nodos|;
8          conj_1 = [nodos(1), nodos(ultimo)]; //Mezcla de los nodos extremos de
           //nodos evaluados
9          conj = maxnodos(codseg(h).nodos, nodos);
10         subconj = minnodos(codseg(h).nodos, nodos);
11         Si subconj ⊆ conj ∧ conj_1 == conj_2
12             swe = true;
13             Adicionar camino etiquetado con j a peneleg;
```

```
14            rutesc = j;
15            tcod = conj(1);
16         Fin Si
17      Fin Si
18  Fin Para
19  Retornar swe, peneleg, rutesc, tcod;
```

b. Verificación de la Ruta a Eliminar: En el Algoritmo 4.13, si la última doble condición en (a) no se cumple (swe desactivado), se procede con el siguiente nodo sumidero en $tdrmin$; por el contrario, si la condición se cumple, se ejecuta el Algoritmo 4.15, donde se procede a dar los siguientes pasos por cada posición j de camino guardada en $peneleg$:

   Se almacena, respectivamente, en las variables $tpen$ (temporal penúltimo) y $tsec$ (temporal segundo nodo), el penúltimo (o precedente del nodo sumidero) y el segundo nodo del $camino$ (posición j del camino guardada en $peneleg$) ubicado en $tdrmin(i).tdrmn$.

   Luego, se busca en qué posición del arreglo $extremos$ está guardado el $tpen$. Es decir, en qué posición l, tal que $tdrmin(i).extremos(l).pen = tpen$. Aquí, la variable i mantiene la posición del sumidero t_i al que está asociado el $camino$ donde se almacena el segundo nodo que, potencialmente, es el punto de partida del camino que anula una solución al sistema lineal de ecuaciones para el grafo multicast con la aplicación de Network Coding. Lo siguiente es encontrar la posición del segundo nodo en la $listseg$ asociada al penúltimo nodo hallado previamente en el arreglo $extremos$.

   Seguidamente, se debe eliminar de la $listseg$, temporalmente, el nodo correspondiente a $tsec$ en la primera posición donde se presente el emparejamiento y, luego guardar la posición de este extremo en el arreglo $posextrmod$, el cual se utilizará para la posterior recuperación y restitución del segundo nodo eliminado. En $posextrmod$ pueden quedar almacenadas tantas posiciones de $extremos$ como elementos estén guardados en $peneleg$.

   Se ejecuta el *xorvect* sobre los segundos nodos almacenados en las $listseg$ reducidas para el sumidero t_i referenciado en $tdrmin(i)$ aplicando la operación descrita en (4.33), donde el símbolo $\bigvee$ representa a la función $xorvect$. Posteriormente, se recupera el segundo nodo eliminado de $listseg$ ($tsec$), a partir de la posición de cada camino a restablecer, guardado en $posextrmod$.

$$zxor = \bigvee_{x=1}^{minflujo} tdrmin(i).extremos(x).listseg \tag{4.33}$$

Algoritmo 4.15: Verificación_Ruta_Eliminar

Entrada: Arreglos $tdrmin, peneleg$ Escalares i
Salida: Arreglo $tdrmin, zxor$

```
1  posextrmod = ∅;
2  Para cada camino etiquetado con j en peneleg
3     ultimo = |tdrmin(i).tdrmn(j).camino|;
```

```
    tpen = tdrmin(i).tdrmn(j).camino(ultimo − 1);
    tsec = tdrmin(i).tdrmn(j).camino(2);
    Sea l la posición tal que tdrmin(i).extremos(l).pen == tpen;
    Sea possec la posición tal que tdrmin(i).extremos(l).listseg(possec) == tsec;
    Eliminar el segundo nodo tdrmin(i).extremos(l).listseg(possec);
    Adicionar l a posextrmod;
Fin Para
zxor = ⋁_{x=1}^{minflujo} tdrmin(i).extremos(x).listseg; //Donde ⋁ = xorvect
Adicionar tsec a tdrmin(i).extremos(l).listseg, ∀l ∈ posextrmod;
Retornar tdrmin, zxor;
```

Según el Algoritmo 4.13, si el resultado de la operación *xorvect* anterior es el conjunto vacío, quiere decir que para los nodos codificadores y segundos nodos agrupados en $codseg(h)$, no es posible eliminar un enlace del camino seleccionado en $tdrmin(i)$, que lleve a una solución con Network Coding cuando se aplique al sistema lineal de ecuaciones derivados del grafo multicast de mínimo flujo máximo G'. Este resultado implica que no se ejecute la revisión con el siguiente sumidero en $tdrmin$ y, por tanto, se desactiva la bandera sw que también controla el ciclo sobre los componentes de $tdrmin$, interrumpiéndolo, y no pudiendo incrementar el contador de iteraciones i sobre $tdrmin$.

Si el *xorvect*, resultado de la operación anterior, es distinto del conjunto vacío, se almacena la posición i del nodo sumidero t_i en el arreglo $sumiseg$ y en la variable $selsum$, y la ruta $rutesc$ en $selrut$ para indicar, que del sumidero t_i, se utilizará la ruta guardada en $selrut$ con el fin de eliminar el enlace que termina en el nodo $tcod$, y que pasa por el segundo nodo $tsec$. Después se continúa con el siguiente nodo sumidero t_{i+1} porque es posible que este o los subsiguientes nodos también presenten la misma situación. Al final de todas las iteraciones sobre el arreglo $tdrmin$, si no se produce ningún conjunto vacío como resultado del *xorvect*, se tendrá almacenado en $selsum$ la posición de $tdrmin$ del último sumidero t_i evaluado y que cumplió con la doble condición en (a); además, se tendrá guardado en $selrut$ la $rutesc$ (ruta seleccionada, de la cual se eliminará el enlace que obstruye la solución) asociada a este último nodo sumidero.

Estos pasos se deben ejecutar cíclicamente para todos los nodos sumideros en $tdrmin$ confrontándolos con los elementos de $codseg(h)$; es decir, para cada t_i cuyo número de caminos después de la mezcla de grafos unicast con mínimo flujo máximo sea mayor que $minflujo$. Si estos pasos se logran para todos los nodos sumideros en esta condición, se tendrá la veracidad para poder corregir la ruta que pasa por el segundo nodo $tsec$, el nodo codificador $tcod$ y cuyo penúltimo nodo es $tpen$, antes de llegar al grupo de nodos sumideros que fueron validados con el *xorvect*.

Si el ciclo de iteraciones sobre $tdrmin$, no se interrumpe por la no desactivación de la bandera sw; es decir, para todos los sumideros t_i que tienen una entrada en $tdrmin$, se completan las iteraciones (las iteraciones se cumplen si el contador de iteraciones i es mayor en 1 que el número de elementos en $tdrmin$, o $i - |tdrmin| = 1$); entonces, se procede a decrementar en 1 cada celda $contarseg(p, q)$, donde $p \in sumiseg$ y q es tal que $segnodtot(q) = tsec$. Si $|contarseg(p)|_{\neq 0} < minflujo$ para cualquier sumidero $p \in sumiseg$, implicaría que no habría un $minflujo$ mínimo de enlaces salientes desde s para cumplir con el mínimo flujo máximo para

todos los sumideros del grafo G' (Lema 4.6), por tanto se mantiene desactivada la bandera swc, se restaura cada celda $contarseg(p,q)$ a su valor original y se vuelve a repetir el ciclo para el siguiente $codseg$ al incrementar el contador h. Si $|contarseg(p)|_{\neq 0} \geq minflujo$ para el primer sumidero $p \in sumiseg$, se activa swc y se detiene el ciclo sobre el arreglo $codseg$, ya que se cumple el Lema 4.6.

Si el ciclo de iteraciones sobre $tdrmin$ se interrumpe por la desactivación de la bandera sw, no se cumplirá la condición $i - |tdrmn| = 1$ y esto conlleva incrementar h para continuar con el siguiente componente de $codseg$. En este caso, la bandera swc sigue desactivada.

Si se activa el swc, se procede a ejecutar la función de Actualización de Flujo (sección 4.3.4), con el fin de efectuar la anulación del camino que pasa por el grupo de nodos: $tsec$, $tcod$, y $tpen$. En caso de mantenerse desactivada swc, se desactiva $swseguir$ para detener un siguiente llamado de la función *opt_minflujo*. Después de desactivar $swseguir$, se revisa el caso especial donde el número de segundos nodos sea $minflujo$ y el arreglo de todas las rutas $tdrmin$ no esté vacío. En caso de cumplirse esta condición, se invoca la función de Verificación de determinantes (sección 4.3.5); si esta devuelve el swp activado, se puede volver a llamar la Actualización de Flujo con los parámetros que devuelve la Verificación de determinantes. En caso de no cumplirse la condición, que el número de segundos nodos es el $minflujo$, se mantiene el grafo multicast G' igual al último que ingresó al Algoritmo 4.13.

Ejemplo 4.24: Continuando con la ejecución de los ejemplos anteriores, se presentarán las dos iteraciones correspondientes a $h = 1$ y $h = 2$, donde h es el contador sobre el cual se itera para $codseg$.

Para $h = 1$, se logra obtener los valores en $peneleg$, $rutesc$ y $tcod$ en la primera iteración sobre j que recorre el arreglo $tdrmin(i).tdrmn$, como se observa en las celdas resaltadas de la Tabla 4.32. Después de $j = 1$, no se logra cambiar más estos valores, dado que no se cumple la doble condición en (a). Como consecuencia de obtener estos tres valores: $peneleg$, $rutesc$ y $tcod$, se procede a ejecutar el paso (b), iterando sobre el número de elementos en $peneleg$ y, dado que solo hay un elemento en este arreglo, el número de iteraciones a ejecutar es solo una; de esta forma se calculan $tpen$ y $tsec$ sobre la única ruta almacenada en $peneleg$ correspondiente al sumidero indexado con $i = 1$. Se ubica dentro de los $extremos$ del $tdrmin(1)$, el $tpen$ y, a la $listseg$ asociada, se le elimina un segundo nodo $tsec$. Esta ubicación corresponde con $tdrmin(1).extremos(3).listseg$, la cual queda modificada como se observa en la Tabla 4.33 en la fila resaltada. El cálculo del *xorvect* sobre los arreglos $listseg$ en la Tabla 4.33, arroja como resultado un conjunto vacío por la reducción al vacío de la $listseg$ correspondiente al $pen = 18$, por ende, la ejecución sobre el resto de nodos sumideros no se ejecuta y, la variable de iteración h de $codseg$, se incrementa para continuar con el siguiente componente de este vector.

Para $h = 2$, se obtiene la secuencia de ejecución mostrada desde la Tabla 4.34 hasta la Tabla 4.43. En la Tabla 4.34, se observa que para $t_i = 23$, en el camino etiquetado con $j = 2$, se logra cumplir la doble condición en (a) y se obtienen los valores para $peneleg$, $rutesc$ y $tcod$ (resaltados en la tabla). Estos valores se mantienen idénticos hasta culminar las iteraciones de los caminos asociados

a $t_i = 23$. Con estos valores se calcula $tpen = tdrmin(1).tdrmn(1).camino(8) = 18$ y $tsec = tdrmin(1).tdrmn(1).camino(2) = 2$. Esto conlleva eliminar un segundo nodo etiquetado con 2 de las $listseg$ asociada al penúltimo nodo 18 del $t_i = 23$ en la Tabla 4.30, quedando modificada como aparece en la Tabla 4.35. En esta tabla, como consecuencia, se observa que el *xorvect* no está vacío y, por ende, se puede determinar el índice del sumidero escogido ($selsum$) y la ruta o camino que conduce a éste ($selrut$), de donde será eliminado el enlace para permitir dar solución al sistema lineal de ecuaciones con el fin de hallar los paquetes simples enviados desde el nodo fuente. Inmediatamente, se continúa con el siguiente nodo sumidero y se repiten estos pasos con $i = 2$ ($t_2 = 24$) e $i = 3$ ($t_3 = 25$). Sin embargo, con $i = 4$ ($t_4 = 26$) e $i = 5$ ($t_5 = 27$), no se logra cumplir con la condición en (a) y, por tanto, no se puede calcular un nuevo $peneleg$, $rutesc$ y $tcod$.

Tabla 4.32 Ejecución del algoritmo en el paso (a) para $h = 1$, $t_1 = 23$

h	$codseg(h)$	$conj_2$
1	[7 15 21 2]	[7 2]

i	t_i
1	23

j	$codint$	$nodos$	$conj_1$	$conj$	$subconj$	$peneleg$	$rutesc$	$tcod$
1	[7 15]	[7 15 2]	[7 2]	[7 15 21 2]	[7 15 2]	[1]	1	7
2	[7 18]	[7 18 2]	[7 2]	[7 15 21 2]	[7 18 2]	[1]	1	7
3	[8 18]	[8 18 2]	[8 2]	[7 15 21 2]	[8 18 2]	[1]	1	7
4	[]					[1]	1	7
5	[7 15]	[7 15 3]	[7 3]	[7 15 21 2]	[7 15 3]	[1]	1	7
6	[7 18]	[7 18 3]	[7 3]	[7 15 21 2]	[7 18 3]	[1]	1	7
7	[8 18]	[8 18 3]	[8 3]	[7 15 21 2]	[8 18 3]	[1]	1	7
8	[]					[1]	1	7
9	[15]	[15 5]	[15 5]	[7 15 21 2]	[15 5]	[1]	1	7

Tabla 4.33 Ejecución del algoritmo en el paso (b) para $h = 1$, $t_1 = 23$

$tpen$	$tsec$
15	2

i	t_i
1	23

pen	$listseg$			
2	2			
13	4			
15	3	5		
18	2	2	3	3
$zxor$	[]			

Tabla 4.34 Ejecución del algoritmo en el paso (a) para $h = 2$, $t_1 = 23$

h	$codseg(h)$	$conj_2$
2	[7 18 2]	[7 2]

i	t_i
1	23

j	$codint$	$nodos$	$conj_1$	$conj$	$subconj$	$peneleg$	$rutesc$	$tcod$
1	[7 15]	[7 15 2]	[7 2]	[7 18 2]	[7 15 2]	[]		
2	[7 18]	[7 18 2]	[7 2]	[7 18 2]	[7 18 2]	[2]	2	7
3	[8 18]	[8 18 2]	[8 2]	[7 18 2]	[8 18 2]	[2]	2	7
4	[]					[2]	2	7
5	[7 15]	[7 15 3]	[7 3]	[7 18 2]	[7 15 3]	[2]	2	7
6	[7 18]	[7 18 3]	[7 3]	[7 18 2]	[7 18 3]	[2]	2	7
7	[8 18]	[8 18 3]	[8 3]	[7 18 2]	[8 18 3]	[2]	2	7
8	[]					[2]	2	7
9	[15]	[15 5]	[15 5]	[7 18 2]	[15 5]	[2]	2	7

Tabla 4.35 Ejecución del algoritmo en el paso (b) para $h = 2, t_1 = 23$

tpen	***tsec***
18	2
i	t_i
1	23

pen	*listseg*			
2	2			
13	4			
15	2	3	5	
18	2	3	3	
zxor	[2 3 4 5]			
selsum	1			
selrut	2			

Tabla 4.36 Ejecución del algoritmo en el paso (a) para $h = 2, t_2 = 24$

h	***codseg(h)***	***conj₂***
2	[7 18 2]	[7 2]

i	t_i	
2	24	

j	***codint***	***nodos***	$conj_1$	***conj***	***subconj***	***peneleg***	***rutesc***	***tcod***
1	[7 15 21]	[7 15 21 2]	[7 2]	[7 15 21 2]	[7 18 2]	[]		
2	[7 18]	[7 18 2]	[7 2]	[7 18 2]	[7 18 2]	[2]	2	7
3	[8 14]	[8 14 2]	[8 2]	[7 18 2]	[8 14 2]	[2]	2	7
4	[8 18]	[8 18 2]	[8 2]	[7 18 2]	[8 18 2]	[2]	2	7
5	[8]	[8 2]	[8 2]	[7 18 2]	[8 2]	[2]	2	7
6	[7 15 21 3]	[7 18 3]	[7 3]	[7 15 21 3]	[7 18 2]	[2]	2	7
7	[7 18]	[7 18 3]	[7 3]	[7 18 2]	[7 18 3]	[2]	2	7
8	[8 14]	[8 14 3]	[8 3]	[7 18 2]	[8 14 3]	[2]	2	7
9	[8 18]	[8 18 3]	[8 3]	[7 18 2]	[8 18 3]	[2]	2	7
10	[8]	[8 3]	[8 3]	[7 18 2]	[8 3]	[2]	2	7
11	[21]	[21 4]	[21 4]	[7 18 2]	[21 4]	[2]	2	7
12	[14]	[14 5]	[14 5]	[7 18 2]	[14 5]	[2]	2	7
13	[15 21]	[15 21 5]	[15 5]	[7 18 2]	[15 21 5]	[2]	2	7

Tabla 4.37 Ejecución del algoritmo en el paso (b) para $h = 2, t_2 = 24$

tpen	***tsec***
18	2
i	t_i
2	24

pen	*listseg*			
8	2	3		
18	2	3	3	
19	2	3	5	
21	2	3	4	5
zxor	[3 4]			
selsum	2			
selrut	2			

Tabla 4.38 Ejecución del algoritmo en el paso (a) para $h = 2, t_3 = 25$

h	***codseg(h)***	***conj₂***
2	[7 18 2]	[7 2]

i	t_i	
3	25	

j	***codint***	***nodos***	$conj_1$	***conj***	***subconj***	***peneleg***	***rutesc***	***tcod***
1	[7 18]	[7 18 2]	[7 2]	[7 18 2]	[7 18 2]	[1]	1	7
2	[8 18]	[8 18 2]	[8 2]	[7 18 2]	[8 18 2]	[1]	1	7
3	[7 18]	[7 18 3]	[7 3]	[7 18 2]	[7 18 3]	[1]	1	7
4	[8 18]	[8 18 3]	[8 3]	[7 18 2]	[8 18 3]	[1]	1	7

5	[]					[1]	1	7
6	[17]	[17 4]	[17 4]	[7 18 2]	[17 4]	[1]	1	7
7	[]					[1]	1	7
8	[17]	[17 6]	[17 6]	[7 18 2]	[17 6]	[1]	1	7

Tabla 4.39 Ejecución del algoritmo en el paso (b) para $h = 2, t_3 = 25$

tpen	***tsec***
18	2
i	t_i
3	25

pen	*listseg*			
16	5			
17	4	6		
18	2	3	3	
20	3			
zxor	[2 3 4 5 6]			
selsum	3			
selrut	1			

Tabla 4.40 Ejecución del algoritmo en el paso (a) para $h = 2, t_4 = 26$

h	***codseg(h)***	***conj₂***
2	[7 18 2]	[7 2]

i	t_i	
4	26	

j	***codint***	***nodos***	$conj_1$	***conj***	***subconj***	***peneleg***	***rutesc***	***tcod***
1	[7 15 21]	[7 15 21 2]	[7 2]	[7 15 21 2]	[7 18 2]	[]		
2	[7 15]	[7 15 2]	[7 2]	[7 18 2]	[7 15 2]	[]		
3	[8 14]	[8 14 2]	[8 2]	[7 18 2]	[8 14 2]	[]		
4	[7 15 21]	[7 15 21 3]	[7 3]	[7 15 21 3]	[7 18 2]	[]		
5	[7 15]	[7 15 3]	[7 3]	[7 18 2]	[7 15 3]	[]		
6	[8 14]	[8 14 3]	[8 3]	[7 18 2]	[8 14 3]	[]		
7	[17]	[17 4]	[17 4]	[7 18 2]	[17 4]	[]		
8	[21]	[21 4]	[21 4]	[7 18 2]	[21 4]	[]		
9	[14]	[14 5]	[7 18 2]	[7 18 2]	[14 5]	[]		
10	[15 21]	[15 21 5]	[15 5]	[7 18 2]	[15 21 5]	[]		
11	[15]	[15 5]	[15 5]	[7 18 2]	[15 5]	[]		
12	[17]	[17 6]	[17 6]	[7 18 2]	[17 6]	[]		

Tabla 4.41 Ejecución del algoritmo en el paso (b) para $h = 2, t_4 = 26$

tpen	***tsec***
i	t_i
4	26

pen	*listseg*			
14	2	3	5	
15	2	3	5	
21	2	3	4	5
22	4	6		
zxor				
selsum	3			
selrut	1			

Tabla 4.42 Ejecución del algoritmo en el paso (a) para $h = 2, t_5 = 27$

h	***codseg(h)***	***conj₂***
2	[7 18 2]	[7 2]

i	t_i	
5	27	

j	***codint***	***nodos***	$conj_1$	***conj***	***subconj***	***peneleg***	***rutesc***	***tcod***

1	[7 15 21]	[7 15 21 2]	[7 2]	[7 15 21 2]	[7 18 2]	[]		
2	[7 15 21]	[7 15 21 3]	[7 3]	[7 15 21 3]	[7 18 2]	[]		
3	[17]	[17 4]	[17 4]	[7 18 2]	[17 4]	[]		
4	[21]	[21 4]	[21 4]	[7 18 2]	[21 4]	[]		
5	[15 21]	[15 21 5]	[15 5]	[7 18 2]	[15 21 5]	[]		
6	[]					[]		
7	[]					[]		
8	[17]	[17 6]	[17 6]	[7 18 2]	[17 6]	[]		

Tabla 4.43 Ejecución del algoritmo en el paso (b) para $h = 2, t_5 = 27$

tpen	***tsec***
i	t_i
5	27

pen	*listseg*			
12	6			
16	5			
17	4	6		
21	2	3	4	5
zxor				

selsum	3
selrut	1

Ejemplo 4.25: Según el Ejemplo 4.24, se cumplió el ciclo de iteraciones sobre $tdrmin$ y, por tanto, la bandera sw no fue desactivada, resultando así el contador i en un valor de 6 ($|tdrmin| + 1$); por tanto, se cumple la condición $i - |tdrmin| = 1$ y, se procede a tomar el camino señalado por el índice $selrut = 1$ del sumidero indicado por $selsum = 3$, donde $t_{selsum} = t_3 = 25$. Este camino corresponde, según la Tabla 4.29 a la secuencia [1 2 7 18 25] y, dado que la variable $tcod = 7$, se toma como nodo predecesor $tnodp = 2$ y, se elimina el enlace (2,7), lo cual equivale a establecer $minmaxflujo(2,7) = 0$, como lo muestra la Figura 4.12. Al eliminar este enlace, el camino anterior desaparece para todos los sumideros t_i que utilizan a $tsec = 2$, $tcod = 7$ y $tpen = 18$ en su trayectoria. Es decir, este camino desaparece para $t_1 = 23, t_2 = 24$ y $t_3 = 25$.

En el arreglo de nodos codificadores $nodos\_cod$, se elimina el $tcod = 7$, resultando en el siguiente nuevo arreglo $nodos\_cod = [8,14,15,17,18,21]$.

Por último, se interrumpe la revisión de los siguientes componentes de $codseg$, la bandera $swseguir$ permanece activa y, se vuelve a ejecutar el algoritmo *opt_minflujo* con un grupo de caminos más reducido.

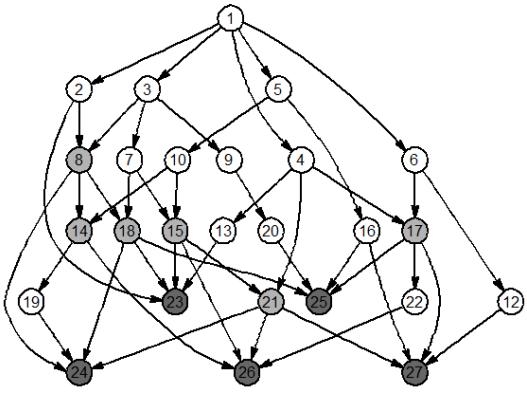


Figura 4.12 Grafo multicast G' después de eliminar el enlace (2,7)

Ejemplo 4.26: Se vuelve a ejecutar el algoritmo *opt_minflujo*, ya que la bandera $swseguir$ continúa activa. Las iteraciones de este algoritmo se llevan a cabo sobre el grafo de la Figura 4.12. El resumen de esta ejecución se presenta desde la Tabla 4.44 hasta la Tabla 4.70 y culmina en la Figura 4.13. En la Tabla 4.44, se presentan los caminos desde el nodo fuente 1 hasta cada uno de los nodos sumideros. En la Tabla 4.45, se presentan los arreglos $extremos$ para cada nodo sumidero. La Tabla 4.46 muestra el resultado de aplicar la creación del arreglo $codseg$. Se puede observar que este arreglo contiene menos entradas. Esto es consecuencia de la eliminación del enlace en la iteración anterior. En la Tabla 4.47, se muestra la primera iteración dentro del arreglo $codseg$ ($h = 1$) y, la Tabla 4.48, muestra que el resultado de la operación *xorvect* es el conjunto vacío, deteniendo la ejecución para $h = 1$ y, conllevando revisar el siguiente $codseg$ ($h = 2$). Los resultados para esta iteración se muestran desde la Tabla 4.49 hasta la Tabla 4.59. En la Tabla 4.55, se observa que el arreglo $peneleg$ =[2,3] contiene más de una ruta (rutas 2 y 3), determinadas por el penúltimo nodo que lleva al nodo sumidero $t_4 = 26$. En cada una de estas rutas correspondientes al $t_4 = 26$, se elimina el segundo nodo antes de calcular el *xorvect*, como lo muestra Tabla 4.56 y Tabla 4.57. En la Tabla 4.57, se calcula el *xorvect* y se obtienen las variables de sumidero seleccionado y ruta seleccionada. Para $t_5 = 27$ con $h = 2$, en la Tabla 4.59, se obtiene un *xorvect* vacío, que interrumpe nuevamente el incremento del contador de iteraciones i para $tdrmin$ y, por ende, no se cumple la condición $i - |tdrmn| = 1$, lo cual conlleva incrementar h y continuar con el contenido del $codseg$ siguiente.

Se continúa con el contenido del $codseg$ para $h = 3$, mostrado desde la Tabla 4.60 hasta la Tabla 4.70. En la Tabla 4.62, se observa que en el $peneleg$ se almacenan las dos rutas (7 y 8) que pasan por los penúltimos nodos 18 y 8, respectivamente. De cada ruta se elimina el segundo nodo especificado por $tsec$, que para ambas rutas es el 3, permitiendo obtener un *xorvect* distinto del conjunto vacío, que conlleva el hallazgo de la ruta $selrut = 8$ dentro del sumidero con índice $selsum = 2$ ($t_2 = 24$). Luego, las iteraciones sobre $tdrmin$ continúan siendo válidas hasta $t_5 = 27$, no desactivando el contador sw, y llevando el contador i hasta el valor de 6; cumpliendo la condición $i - |tdrmin| = 1$. Se toma el valor del sumidero seleccionado ($selsum = 3$) y la ruta seleccionada ($selrut = 3$) dentro de este, indicados en la Tabla 4.70 para actualizar la ruta [1 3 8 18 25] mostrada en la Tabla 4.44. Se toma el valor del último $tcod = 8$ para determinar el nodo predecesor $tnodp = 3$ y así eliminar el enlace (3,8), o lo que es lo mismo establecer $minmaxflujo(3,8) = 0$, como lo muestra la Figura 4.13. Al eliminar este enlace, el camino anterior desaparece para todos los sumideros t_i que utilizan a $tsec = 3$, $tcod = 8$ y $tpen = 18$ en su trayectoria. Es decir, este camino desaparece para $t_1 = 23, t_2 = 24$ y $t_3 = 25$.

En el arreglo de nodos codificadores $nodos\_cod$, se elimina el $tcod = 8$, resultando en el siguiente nuevo arreglo $nodos\_cod = [14,15,17,18,21]$.

Por último, se interrumpe la revisión de los siguientes componentes de $codseg$, la bandera $swseguir$ permanece activa y, se vuelve a ejecutar el algoritmo *opt_minflujo* con un grupo de caminos más reducido; sin embargo, después de esta tercera ejecución, el arreglo $codseg$ no se logra construir y permanece vacío, devolviendo desactivada la bandera $swseguir$. Esto conlleva obtener en forma definitiva el grafo multicast de mínimo flujo máximo reducido, mostrado en la Figura 4.13, para entrega de paquetes utilizando el paradigma de Network Coding.

Tabla 4.44 Caminos que llegan a cada sumidero desde el nodo fuente en G' de la Figura 4.12

Caminos de 1 a 23				
1	2	8	18	23
1	2	23		
1	3	7	15	23
1	3	7	18	23
1	3	8	18	23
1	4	13	23	
1	5	10	15	23

Caminos de 1 a 24					
1	2	8	14	19	24
1	2	8	18	24	
1	2	8	24		
1	3	7	15	21	24
1	3	7	18	24	
1	3	8	14	19	24
1	3	8	18	24	
1	3	8	24		
1	4	21	24		
1	5	10	14	19	24
1	5	10	15	21	24

Caminos de 1 a 25				
1	2	8	18	25
1	3	7	18	25
1	3	8	18	25
1	3	9	20	25
1	4	17	25	
1	5	16	25	
1	6	17	25	

Caminos de 1 a 26					
1	2	8	14	26	
1	3	7	15	21	26
1	3	7	15	26	
1	3	8	14	26	
1	4	17	22	26	
1	4	21	26		
1	5	10	14	26	
1	5	10	15	21	26
1	5	10	15	26	
1	6	17	22	26	

Caminos de 1 a 27					
1	3	7	15	21	27
1	4	17	27		
1	4	21	27		
1	5	10	15	21	27
1	5	16	27		
1	6	12	27		
1	6	17	27		

Tabla 4.45 $tdrmin(i).extremos$ para cada nodo sumidero

i	t_i	i	t_i	i	t_i	i	t_i	i	t_i
1	**23**	**2**	**24**	**3**	**25**	**4**	**26**	**5**	**27**
pen	$listseg$	pen	$listseg$	pen	$listseg$	pen	$listseg$	pen	$listseg$
2	2	8	2 3	16	5	14	2 3 5	12	6
13	4	18	2 3 3	17	4 6	15	3 5	16	5
15	3 5	19	2 3 5	18	2 3 3	21	3 4 5	17	4 6
18	2 3 3	21	3 4 5	20	3	22	4 6	21	3 4 5

Tabla 4.46 Arreglo $codseg$ para segunda iteración con G' de 27 nodos

cod		*seg*
8	18	2
15		3
8	18	3
15		5

Tabla 4.47 Ejecución del algoritmo en el paso (a) para $h = 1, t_1 = 23$

h	$codseg(h)$	$conj_2$
1	[8 18 2]	[8 2]

i	t_i
1	23

j	$codint$	$nodos$	$conj_1$	$conj$	$subconj$	$peneleg$	$rutesc$	$tcod$
1	[8 18]	[8 18 2]	[8 2]	[8 18 2]	[8 18 2]	[1]	1	8
2	[]					[1]	1	8
3	[15]	[15 3]	[15 3]	[8 18 2]	[15 3]	[1]	1	8
4	[18]	[18 3]	[18 3]	[8 18 2]	[18 3]	[1]	1	8
5	[8 18]	[8 18 3]	[18 3]	[8 18 2]	[8 18 3]	[1]	1	8
6	[]					[1]	1	8
7	[15]	[15 5]	[15 5]	[8 18 2]	[15 5]	[1]	1	8

Tabla 4.48 Ejecución del algoritmo en el paso (b) para $h = 1, t_1 = 23$

tpen	***tsec***
18	2
i	t_i
1	23

pen	*listseg*			
2	2			
13	4			
15	3	5		
18	3	3		
zxor	[]			

Tabla 4.49 Ejecución del algoritmo en el paso (a) para $h = 2, t_1 = 23$

h	***codseg(h)***	***conj$_2$***
2	[15 3]	[15 3]

i	t_i
1	23

j	***codint***	***nodos***	***conj$_1$***	***conj***	***subconj***	***peneleg***	***rutesc***	***tcod***
1	[8 18]	[8 18 2]	[8 2]	[8 18 2]	[15 3]	[]		
2	[]					[]		
3	[15]	[15 3]	[15 3]	[15 3]	[15 3]	[3]	3	15
4	[18]	[18 3]	[18 3]	[15 3]	[18 3]	[3]	3	15
5	[8 18]	[8 18 3]	[8 3]	[8 18 3]	[15 3]	[3]	3	15
6	[]					[3]	3	15
7	[15]	[15 5]	[15 5]	[15 3]	[15 5]	[3]	3	15

Tabla 4.50 Ejecución del algoritmo en el paso (b) para $h = 2, t_1 = 23$

tpen	***tsec***
15	3
i	t_i
1	23

pen	*listseg*			
2	2			
13	4			
15	5			
18	2	3	3	
zxor	[4 5]			
selsum	1			
selrut	3			

Tabla 4.51 Ejecución del algoritmo en el paso (a) para $h = 2, t_2 = 24$

h	***codseg(h)***	***conj$_2$***
2	[15 3]	[15 3]

i	t_i
2	24

j	***codint***	***nodos***	***conj$_1$***	***conj***	***subconj***	***peneleg***	***rutesc***	***tcod***
1	[8 14]	[8 14 2]	[8 2]	[8 14 2]	[15 3]	[]		
2	[8 18]	[8 18 2]	[8 2]	[8 18 2]	[15 3]	[]		
3	[8]	[8 2]	[8 2]	[15 3]	[8 2]	[]		
4	[15 21]	[15 21 3]	[15 3]	[15 21 3]	[15 3]	[4]	4	15
5	[18]	[18 3]	[18 3]	[15 3]	[18 3]	[4]	4	15
6	[8 14]	[8 14 3]	[8 3]	[8 14 3]	[15 3]	[4]	4	15
7	[8 18]	[8 18 3]	[8 3]	[8 18 3]	[15 3]	[4]	4	15
8	[8]	[8 3]	[8 3]	[15 3]	[8 3]	[4]	4	15
9	[21]	[21 4]	[21 4]	[15 3]	[21 4]	[4]	4	15
10	[14]	[14 5]	[14 5]	[15 3]	[14 5]	[4]	4	15
11	[15 21]	[15 21 5]	[15 5]	[15 21 5]	[15 3]	[4]	4	15

Tabla 4.52 Ejecución del algoritmo en el paso (b) para $h = 2, t_2 = 24$

tpen					***tsec***
21					3
i					t_i
2					24
pen	*listseg*				
8	2	3			
18	2	3	3		
19	2	3	5		
21	4	5			
zxor	[2 4]				
selsum					2
selrut					4

Tabla 4.53 Ejecución del algoritmo en el paso (a) para $h = 2, t_3 = 25$

h	***codseg(h)***	***conj₂***
2	[15 3]	[15 3]

i	t_i	
3	25	

j	***codint***	***nodos***	***conj₁***	***conj***	***subconj***	***peneleg***	***rutesc***	***tcod***
1	[8 18]	[8 18 2]	[8 2]	[8 18 2]	[15 3]	[]		
2	[18]	[18 3]	[18 3]	[15 3]	[18 3]	[]		
3	[8 18]	[8 18 3]	[8 3]	[8 18 3]	[15 3]	[]		
4	[]					[]		
5	[17]	[17 4]	[17 4]	[15 3]	[17 4]	[]		
6	[]					[]		
7	[17]	[17 6]	[17 6]	[15 3]	[17 6]	[]		

Tabla 4.54 Ejecución del algoritmo en el paso (b) para $h = 2, t_3 = 25$

tpen					***tsec***
i					t_i
3					25
pen	*listseg*				
16	5				
17	4	6			
18	2	3	3		
20	3				
zxor					
selsum					2
selrut					4

Tabla 4.55 Ejecución del algoritmo en el paso (a) para $h = 2, t_4 = 26$

h	***codseg(h)***	***conj₂***
2	[15 3]	[15 3]

i	t_i	
4	26	

j	***codint***	***nodos***	***conj₁***	***conj***	***subconj***	***peneleg***	***rutesc***	***tcod***
1	[8 14]	[8 14 2]	[8 2]	[8 14 2]	[15 3]	[]		
2	[15 21]	[15 21 3]	[15 3]	[15 21 3]	[15 3]	[2]	2	15
3	[15]	[15 3]	[15 3]	[15 3]	[15 3]	[2 3]	3	15
4	[8 14]	[8 14 3]	[8 3]	[8 14 3]	[15 3]	[2 3]	3	15
5	[17]	[17 4]	[17 4]	[15 3]	[17 4]	[2 3]	3	15
6	[21]	[21 4]	[21 4]	[15 3]	[21 4]	[2 3]	3	15
7	[14]	[14 5]	[14 5]	[15 3]	[14 5]	[2 3]	3	15
8	[15 21]	[15 21 5]	[15 5]	[15 21 5]	[15 3]	[2 3]	3	15
9	[15]	[15 5]	[15 5]	[15 3]	[15 5]	[2 3]	3	15
10	[17]	[17 6]	[17 6]	[15 3]	[17 6]	[2 3]	3	15

Tabla 4.56 Ejecución del algoritmo en el paso (b) para $h = 2, t_4 = 26, peneleg(1) = 2$

tpen					***tsec***
21					3
i					t_i
4					26
pen	*listseg*				
14	2	3	5		
15	3	5			
21	4	5			
22	4	6			
zxor					
selsum					
selrut					

Tabla 4.57 Ejecución del algoritmo en el paso (b) para $h = 2, t_4 = 26, peneleg(2) = 3$

tpen					***tsec***
15					3
i					t_i
4					26
pen	*listseg*				
14	2	3	5		
15	5				
21	4	5			
22	4	6			
zxor	[2 3 5 6]				
selsum	4				
selrut	3				

Tabla 4.58 Ejecución del algoritmo en el paso (a) para $h = 2, t_5 = 27$

h	***codseg(h)***	***conj₂***
2	[15 3]	[15 3]

i	t_i	
5	27	

j	***codint***	***nodos***	$conj_1$	***conj***	***subconj***	***peneleg***	***rutesc***	***tcod***
1	[15 21]	[15 21 3]	[15 3]	[15 21 3]	[15 3]	[1]	1	15
2	[17]	[17 4]	[17 4]	[15 3]	[17 4]	[1]	1	15
3	[21]	[21 4]	[21 4]	[15 3]	[21 4]	[1]	1	15
4	[15 21]	[15 21 5]	[15 5]	[15 21 5]	[15 3]	[1]	1	15
5	[]					[1]	1	15
6	[]					[1]	1	15
7	[17]	[17 6]	[17 6]	[15 3]	[17 6]	[1]	1	15

Tabla 4.59 Ejecución del algoritmo en el paso (b) para $h = 2, t_5 = 27$

tpen					***tsec***
21					3
i					t_i
5					27
pen	*listseg*				
12	6				
16	5				
17	4	6			
21	4	5			
zxor	[]				
selsum	3				
selrut	1				

Tabla 4.60 Ejecución del algoritmo en el paso (a) para $h = 3, t_1 = 23$

h	$codseg(h)$	$conj_2$
3	[8 18 3]	[8 3]

i	t_i
1	23

j	$codint$	$nodos$	$conj_1$	$conj$	$subconj$	$peneleg$	$rutesc$	$tcod$
1	[8 18]	[8 18 2]	[8 2]	[8 18 3]	[8 18 2]	[]		
2	[]					[]		
3	[15]	[15 3]	[15 3]	[8 18 3]	[15 3]	[]		
4	[18]	[18 3]	[18 3]	[8 18 3]	[18 3]	[]		
5	[8 18]	[8 18 3]	[8 3]	[8 18 3]	[8 18 3]	[5]	5	8
6	[]					[5]	5	8
7	[15]	[15 5]	[15 5]	[8 18 3]	[15 5]	[5]	5	8

Tabla 4.61 Ejecución del algoritmo en el paso (b) para $h = 3, t_1 = 23$

$tpen$	$tsec$
18	3
i	t_i
1	23

pen	$listseg$			
2	2			
13	4			
15	3	5		
18	2	3		
$zxor$	[4 5]			
$selsum$	1			
$selrut$	5			

Tabla 4.62 Ejecución del algoritmo en el paso (a) para $h = 3, t_2 = 24$

h	$codseg(h)$	$conj_2$
3	[8 18 3]	[8 3]

i	t_i
2	24

j	$codint$	$nodos$	$conj_1$	$conj$	$subconj$	$peneleg$	$rutesc$	$tcod$
1	[8 14]	[8 14 2]	[8 2]	[8 18 3]	[8 14 2]	[]		
2	[8 18]	[8 18 2]	[8 2]	[8 18 3]	[8 18 2]	[]		
3	[8]	[8 2]	[8 2]	[8 18 3]	[8 2]	[]		
4	[15 21]	[15 21 3]	[15 3]	[8 18 3]	[15 21 3]	[]		
5	[18]	[18 3]	[18 3]	[8 18 3]	[18 3]	[]		
6	[8 14]	[8 14 3]	[8 3]	[8 18 3]	[8 14 3]	[]		
7	[8 18]	[8 18 3]	[8 3]	[8 18 3]	[8 18 3]	[7]	7	8
8	[8]	[8 3]	[8 3]	[8 18 3]	[8 3]	[7 8]	8	8
9	[21]	[21 4]	[21 4]	[8 18 3]	[21 4]	[7 8]	8	8
10	[14]	[14 5]	[14 5]	[8 18 3]	[14 5]	[7 8]	8	8
11	[15 21]	[15 21 5]	[15 5]	[8 18 3]	[15 21 5]	[7 8]	8	8

Tabla 4.63 Ejecución del algoritmo en el paso (b) para $h = 3, t_2 = 24,\ peneleg(1) = 7$

$tpen$	$tsec$
18	3
i	t_i
2	24

pen	$listseg$			
8	2	3		
18	2	3		
19	2	3	5	
21	3	4	5	
$zxor$				
$selsum$				

selrut	

Tabla 4.64 Ejecución del algoritmo en el paso (b) para $h = 3, t_2 = 24,\ peneleg(2) = 8$

tpen					***tsec***
8					3
i					t_i
2					24
pen	*listseg*				
8	2				
18	2	3			
19	2	3	5		
21	3	4	5		
zxor	[2 3 4]				
selsum	2				
selrut	8				

Tabla 4.65 Ejecución del algoritmo en el paso (a) para $h = 3, t_3 = 25$

h	***codseg(h)***		***conj₂***					
3	[8 18 3]		[8 3]					
	i	t_i						
	3	25						
	j	***codint***	***nodos***	***conj₁***	***conj***	***subconj***	***peneleg***	***rutesc***	***tcod***
	1	[8 18]	[8 18 2]	[8 2]	[8 18 3]	[8 18 2]	[]		
	2	[18]	[18 3]	[18 3]	[8 18 3]	[18 3]	[]		
	3	[8 18]	[8 18 3]	[8 3]	[8 18 3]	[8 18 3]	[3]	3	8
	4	[]					[3]	3	8
	5	[17]	[17 4]	[17 4]	[8 18 3]	[17 4]	[3]	3	8
	6	[]					[3]	3	8
	7	[17]	[17 6]	[17 6]	[8 18 3]	[17 6]	[3]	3	8

Tabla 4.66 Ejecución del algoritmo en el paso (b) para $h = 3, t_3 = 25$

tpen					***tsec***
18					3
i					t_i
3					25
pen	*listseg*				
16	5				
17	4	6			
18	2	3			
20	3				
zxor	[2 4 5 6]				
selsum	3				
selrut	3				

Tabla 4.67 Ejecución del algoritmo en el paso (a) para $h = 3, t_4 = 26$

h	***codseg(h)***		***conj₂***					
3	[8 18 3]		[8 3]					
	i	t_i						
	4	26						
	j	***codint***	***nodos***	***conj₁***	***conj***	***subconj***	***peneleg***	***rutesc***	***tcod***
	1	[8 14]	[8 14 2]	[8 2]	[8 18 3]	[8 14 2]	[]		
	2	[15 21]	[15 21 3]	[15 3]	[8 18 3]	[15 21 3]	[]		
	3	[15]	[15 3]	[15 3]	[8 18 3]	[15 3]	[]		
	4	[8 14]	[8 14 3]	[8 3]	[8 18 3]	[8 14 3]	[]		
	5	[17]	[17 4]	[17 4]	[8 18 3]	[17 4]	[]		
	6	[21]	[21 4]	[21 4]	[8 18 3]	[21 4]	[]		
	7	[14]	[14 5]	[14 5]	[8 18 3]	[14 5]	[]		
	8	[15 21]	[15 21 5]	[15 5]	[8 18 3]	[15 21 5]	[]		

9	[15]	[15 5]	[15 5]	[8 18 3]	[15 5]	[]		
10	[17]	[17 6]	[17 6]	[8 18 3]	[17 6]	[]		

Tabla 4.68 Ejecución del algoritmo en el paso (b) para $h = 3$, $t_4 = 26$

tpen					***tsec***
i					t_i
4					26
pen	*listseg*				
14	2	3	5		
15	3	5			
21	3	4	5		
22	4	6			
zxor					
selsum					3
selrut					3

Tabla 4.69 Ejecución del algoritmo en el paso (a) para $h = 3$, $t_5 = 27$

h	***codseg(h)***		***conj$_2$***					
3	[8 18 3]		[8 3]					
	i	t_i						
	5	27						
	j	***codint***	***nodos***	***conj$_1$***	***conj***	***subconj***	***peneleg***	***rutesc***	***tcod***
	1	[15 21]	[15 21 3]	[15 3]	[8 18 3]	[15 21 3]	[]		
	2	[17]	[17 4]	[17 4]	[8 18 3]	[17 4]	[]		
	3	[21]	[21 4]	[21 4]	[8 18 3]	[21 4]	[]		
	4	[15 21]	[15 21 5]	[15 5]	[8 18 3]	[15 21 5]	[]		
	5	[]					[]		
	6	[]					[]		
	7	[17]	[17 6]	[17 6]	[8 18 3]	[17 6]	[]		

Tabla 4.70 Ejecución del algoritmo en el paso (b) para $h = 3$, $t_5 = 27$

tpen					***tsec***
i					t_i
5					27
pen	*listseg*				
12	6				
16	5				
17	4	6			
21	3	4	5		
zxor					
selsum					3
selrut					3

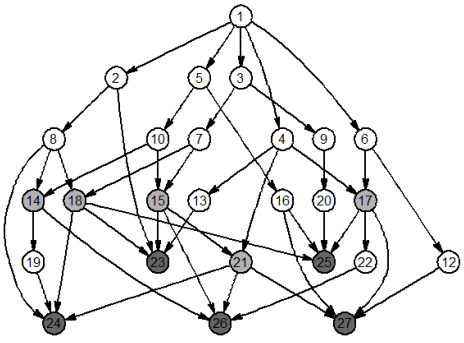


Figura 4.13 Grafo multicast G' después de eliminar el enlace (3,8)

4.3.2 Algoritmo xor de conjuntos de segundos nodos:xorvect

El objetivo de este procedimiento es tomar dos arreglos de segundos nodos y determinar si es posible aplicar el xor con ellos (o la operación diferencia simétrica de conjuntos (sección 2.1.3) sobre ellos), de tal forma que permita sobre el grafo multicast G', resolver el sistema lineal de ecuaciones sobre la base de Network Coding, generando en cada nodo sumidero los paquetes originales enviados desde el nodo fuente. La función *xorvect* (Algoritmo 4.16) se ejecuta de la siguiente forma:

a. Recibe dos vectores ($listseg$), cada uno conteniendo los segundos nodos $seg_m, m \leq |listseg|$ de los caminos que convergen en un penúltimo nodo o predecesor $p_j, j \leq minflujo$ de un sumidero t_i. Es decir, cada segundo nodo representa un paquete diferente emitido por el nodo fuente. La primera acción a ejecutar es la aplicación de la función *reducir* (sección 4.3.3) a cada conjunto de segundos nodos.

   Si en una $listseg$ existen nodos repetidos seg_{m_1} y seg_{m_2}, quiere decir que dos caminos distintos m_1 y m_2 están alimentando o contribuyendo al penúltimo nodo (predecesor del nodo sumidero) p_j con el mismo paquete, estos se anulan según la reglas de ejecución de la función *reducir*. Si la resultante es un arreglo vacío, no se puede obtener una solución al sistema lineal de ecuaciones en el grafo multicast. Esto conlleva que en el algoritmo *opt_minflujo*, esta alternativa de solución se deba descartar, dado que está informando que por un mismo enlace entrante (x, p_j) al nodo p_j, todo el flujo de paquetes que ingrese, se anule. Consecuentemente, imposibilitaría una solución, dado que el enlace (p_j, t_i) no transportaría ningún paquete al sumidero t_i y, por ende, el mínimo flujo máximo ($minflujo$) para este nodo se disminuiría en 1, no logrando reunir las combinaciones lineales necesarias para obtener la solución al sistema y, por tanto, la no decodificación de los paquetes originales.

   Ejemplo 4.27: La Figura 4.14 muestra que al nodo 18, el cual es penúltimo nodo de los sumideros 23, 24 y 25, ingresa el paquete combinado A+B en forma repetida, por los enlaces (8,18) y (7,18). Esto significa que la lista de segundos nodos de 18 corresponde con el arreglo $listseg_{18} = [2,2,3,3]$, donde el nodo 2 es el segundo nodo por donde se transfiere A, y 3 es el segundo nodo por donde se transfiere B. La función $reducir$ anularía esta lista y, por ende, la función *xorvect* devolvería como resultante un conjunto vacío y la bandera de control en *falso*. En este caso no se alcanzaría a realizar la diferencia de los conjuntos de segundos nodos asociados a otros penúltimos nodos de los sumideros 23, 24 y 25, con el conjunto de segundos nodos de 18.

b. Si las dos $listseg$ recibidas y reducidas no son vacías, al aplicar la función xor de conjuntos (diferencia simétrica de conjuntos) sobre estos dos arreglos, se puede obtener un conjunto vacío como respuesta a retornar y, por ende, la bandera de control en *falso* nuevamente. En este caso, se efectúa el xor de dos conjuntos $listseg_{i_1}$ y $listseg_{i_2}$ reducidos no vacíos para un mismo sumidero t_i, pero con los mismos segundos nodos. Esto conlleva generar un conjunto vacío. La segunda posibilidad es que se genere como resultante un conjunto no vacío y la bandera de control se establece en *verdadero*.

Ejemplo 4.28: La Figura 4.8 muestra como al nodo sumidero 26 ingresa en forma duplicada el paquete combinado A+B+C por los enlaces (14,26) y (15,26), y que corresponde a las listas de segundos nodos $listseg_{14} = listseg_{15} = [2,3,5]$. Al aplicar la función *xorvect* sobre estas, devolverá como resultante un conjunto vacío y la bandera de control en *falso*.

c. Si el *xorvect* obtenido no está vacío, se supondría que se tiene una solución al sistema de ecuaciones para este nodo sumidero en un grafo multicast. Si el *xorvect* está vacío, no se podrá corregir el grafo multicast con este sumidero.

Algoritmo 4.16: *xorvect* : xor de segundos nodos

Entrada: Arreglos a, b
Salida: Arreglo $xorvect$ Escalar sw

```
1  ar = reducir(a);
2  br = reducir(b);
3  Si ar == ∅ ∨ br == ∅
4     xorvect = ∅;
5  Sino
6     xorvect = ar Δ br;
7  Fin Si
8  Si xorvect == ∅
9     sw = false;
10 Sino
11    sw = true;
12 Fin Si
13 Retornar xorvect,sw;
```

4.3.3 Algoritmo de reducción de conjuntos de segundos nodos: reducir

Como resultado de la mezcla de los grafos unicast de flujo máximo individuales ya depurados, a un penúltimo nodo (o predecesor de nodo sumidero), le puede llegar un flujo resultante de paquetes que sea la sumatoria xor de varios caminos que pasan por el mismo segundo nodo. La Figura 4.14 muestra que en el nodo 18, el cual es el penúltimo nodo antes de llegar al nodo sumidero 23, convergen dos flujos provenientes de los segundos nodos 2 y 3. Ante la presencia de los nodos codificadores 7 y 8, que son alimentados simultáneamente por los enlaces que se desprenden de estos segundos nodos, se configura un mismo paquete combinado A+B que ingresará tanto por el enlace (8,18) y (7,18) y se anulará en 18, que actúa como codificador. Por tanto, el nodo 18 no entregará nada al nodo 23. En conclusión, el nodo 18 recibe el conjunto de paquetes [A,A,B,B], que en términos de segundos nodos corresponde al arreglo [2,2,3,3].

La función *reducir*, basada en la Definición 4.5, tiene como objetivo reducir los segundos nodos repetidos bajo una operación que funcione con la suma módulo 2. Es decir, si existe un número impar de nodos repetidos, solo se mantiene uno, y si el número de nodos repetidos es par, todos se eliminan y se obtiene un conjunto vacío. En el ejemplo descrito, según la Figura 4.14, se anularía toda la lista y el resultado estaría vacío.

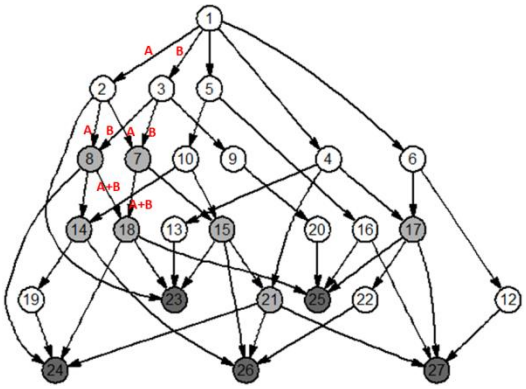


Figura 4.14 Anulación de flujos en penúltimo nodo 18

4.3.4 Algoritmo de actualización de flujo: Actflujo

El Algoritmo 4.17 recibe como entradas la matriz que corresponde al grafo multicast de flujo máximo, $minmaxflujo$; el arreglo de nodos codificadores, $nodos\_cod$; el camino correspondiente a $tdrmin(selsum).tdrmn(selrut).camino$; y el nodo codificador, $tcod$, por el cual pasa este camino y que es el objetivo al que se eliminará un enlace entrante.

Para cumplir la meta de actualización del camino, se busca el nodo predecesor de $tcod$. Sea $tnodp$ el nodo predecesor de $tcod$ en esta ruta; luego, se elimina este camino desde $tsec$ hasta $tpen$, pasando por $tcod$, y anulando el enlace $(tnodp, tcod)$. Es decir, $minmaxflujo(tnodp, tcod) = 0$. Como consecuencia de la eliminación del enlace entrante al nodo codificador, se debe revisar si no ingresan, aún, más de un enlace a este nodo; de ser así, el nodo deja de ser codificador y se elimina de la lista $nodos\_cod$.

El algoritmo también elimina caminos que no conducen a ningún nodo sumidero y que se pueden considerar espurios, dado que no finalizan en ningún nodo sumidero.

Algoritmo 4.17: Actflujo : Actualización de flujo

Entrada: Arreglos $minmaxflujo, nodos\_cod, camino$ Escalar $tcod$
Salida: Arreglo $minmaxflujo, nodos\_cod$

```
1  Hallar un selnod tal que camino(selnod) == tcod;
2  tnodp = camino(selnod − 1);
3  minmaxflujo(tnodp, tcod) = 0;
4  Si |minmaxflujo(tnodp)|≠0 == 0 // Es un camino espurio a partir de tnodp
5      ∀ enlace (ip, tdnop) ∈ G', minmaxflujo(ip, tnodp) = 0; //Eliminar enlaces
                                                              //previos a tnodp
6  Fin Si
7  Si |minmaxflujo^T(tcod)|=1 < 2 //Actualización de nodos_cod
8      nodos_cod(pcod) = ∅, tal que nodos_cod(pcod) == tcod;
9  Fin Si
10 Retornar minmaxflujo, nodos_cod;
```

4.3.5 Algoritmo de verificación de determinantes: Verdet

Esta función, en particular, se utiliza cuando se presenta un número de segundos nodos igual a $minflujo$ y, dado que el número de penúltimos nodos (o nodos predecesores) de cada sumidero es igual a $minflujo$ (Lema 4.2), se puede generar una matriz cuadrada que muestre cuantas veces contribuye cada segundo nodo en los nodos predecesores de cada sumidero antes que este reciba los paquetes entrantes (simples o combinados) de forma definitiva.

Sea κ el número de enlaces de salida desde el nodo fuente s (κ es el número de segundos nodos), y r el mínimo flujo máximo común del grafo multicast unisesión. Si $r = \kappa$, se infiere que por cada uno de los κ enlaces salientes de s, debe emerger uno y solo uno de los r-paquetes que conforman el flujo máximo común. A su vez, cada uno de los r-penúltimos nodos tiene asociado el conjunto de segundos nodos que determinan las combinaciones lineales de paquetes que entregan a los sumideros que preceden. Por tanto, de los r penúltimos nodos predecesores de cada sumidero $t_i \in T$, deben emerger enlaces de llegada que entreguen paquetes (simples o combinados) que formen un sistema linealmente independiente o que la matriz con elementos en el campo $\mathbb{F}_2$ que constituyan, sea invertible.

Por cada nodo sumidero t_i, se constituirá una matriz $supmat(i)$ cuadrada de dimensión $r \times r$, como lo muestra la Figura 4.15, donde las r filas corresponden al número de penúltimo nodos que preceden al sumidero t_i y, las r columnas están asociadas al número de segundos nodos en que finalizan los enlaces de salida desde el nodo fuente s. Se construye la matriz $supmat(i)$ por cada sumidero t_i, donde se indique la contribución de cada paquete emergente en s y que pasa por un segundo nodo seg_k, ($seg_k = segnodtot(k)$), en la combinación lineal (paquete) que arriba a cada penúltimo nodo p_i^j, $(1 \le i \le |T|, 1 \le j \le r)$ de t_i. Se parte de la lista de segundos nodos de cada penúltimo nodo, $listseg(p_i^j)$, para hallar cuántas veces un segundo nodo contribuye en las combinaciones lineales que se forman en p_i^j.

$$\begin{array}{c} \\ p_i^1 \\ p_i^2 \\ \vdots \\ p_i^r \end{array} \begin{array}{c} \begin{array}{cccc} seg_1 & seg_1 & \dots & seg_r \end{array} \\ \begin{bmatrix} 0 & 0 & \dots & 1 \\ 1 & 3 & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 2 & 0 & \dots & 4 \end{bmatrix} \end{array}$$

Figura 4.15 Forma de la matriz $supmat(i)$ para el sumidero t_i

Inicialmente, la matriz $supmat(i)$ se inicializa con cero en cada posición, y el resultado de la contribución de cada segundo nodo a la combinación lineal que se forma en los penúltimos nodos que preceden al sumidero t_i, se calcula según (4.34):

$$supmat(i,j,k) = \begin{cases} supmat(i,j,k) + +, \text{si } seg \in listseg\left(p_i^j\right) \text{ y } k\text{: } seg = segnodtot(k) \\ 0, \text{en otro caso} \end{cases} \quad (4.34)$$

Cada posición $supmat(i,j,k) = c$, significa que al penúltimo nodo p_i^j, asociado al sumidero t_i, ingresan c-caminos que han pasado por el segundo nodo seg_k. La matriz $msupmat$ redefine a la matriz $supmat$, como una matriz con elementos en el campo $\mathbb{F}_2$, según (4.35):

$$msupmat(i,j,k) = supmat(i,j,k) \bmod 2 \tag{4.35}$$

Si el número de contribuciones de un segundo nodo seg_k, a través de los caminos que ingresan al penúltimo nodo p_i^j del nodo sumidero t_i, es impar, se infiere que, realmente llegará el paquete que se envía a través del enlace de salida (s, seg_k) al sumidero. Este paquete puede llegar de forma individual o combinado con otros paquetes. Por el contrario, si el número de contribuciones del segundo nodo seg_k es par, se infiere que no llegará el paquete que se envía a través del enlace de salida (s, seg_k) al sumidero.

La política del algoritmo consiste, para cada nodo sumidero t_i, en examinar la posibilidad de reajustar un camino que pase por un conjunto de nodos codificadores, de tal forma que se elimine la ruta que va desde el nodo s hasta el primer nodo del conjunto de codificadores, y que atraviese por un segundo nodo específico seg_k. Se determinan los penúltimos nodos p_i^j, que tienen el segundo nodo seg_k en su lista, para actualizar las entradas correspondientes a sus posiciones en la matriz $msupmat$ mediante (4.36). Luego, si cada $msupmat(i), 1 \leq i \leq |T|$, es no singular, se procede a eliminar el enlace que finaliza en el nodo codificador y que hace parte de la ruta que contiene al segundo nodo seg_k.

$$msupmat(i,j,k) = (msupmat(i,j,k) + 1) \bmod 2 \tag{4.36}$$

A continuación se presenta el Algoritmo 4.18 que realiza los siguientes pasos:

a. Inicialización de la matriz $supmat(i)$ para cada $t_i \in T$ con valores nulos, además de iniciar con valor vacío las variables $tcod, selsum$ y $selrut$.
b. Incrementar en 1 $supmat(i,j,k)$ para cada $t_i \in T$, de tal forma que para cada penúltimo nodo p_i^j que preceda a t_i, se halle el segundo nodo seg_k donde $tdrmin(i).extremos(j).listseg(l) = seg_k, \forall l = 1, \dots, |tdrmin(i).extremos(j).listseg|$.
c. Cálculo de $msupmat$, a partir de la matriz $supmat$, según (4.35).
d. Cálculo del determinante $dsupm(i)$ para cada matriz $msupmat(i)$.
e. Si todos los determinantes en $dsupm$ son no nulos, la comprobación de determinantes está correcta, el sw es activado y finaliza la comprobación. Sino, el sw es desactivado y se continúa con los siguientes pasos.
f. Si el sw está desactivado, se calcula el arreglo $codseg$ invocando la función Conscodseg (Algoritmo 4.12).
g. Se ejecutan los siguientes tres pasos para cada lista de nodos codificadores y segundo nodo en cada posición $h \leq |codseg|$, controlados también por la bandera sw que se mantendrá inactiva (false) mientras no se cumpla que los determinantes para cada sumidero t_i sean distintos de nulo.
h. Para cada $codseg$ se almacena temporalmente $msupmat$ y $dsupm$ en $tsupmat$ y $tdsupm$, respectivamente y, se desactiva la bandera swt. Se itera con la variable j para recorrer y revisar

cada $tdrmin$, controlada con la bandera swt. Dentro de cada iteración sobre $tdrmin$ se itera con la variable k sobre $tdrmin(j).tdrmn$ para determinar los caminos que cumplen que la concatenación de sus nodos codificadores (si es diferente de vacío) y su segundo nodo, es un subconjunto de los nodos codificadores y segundo nodo almacenados en $codseg(h).nodos$. Si se cumple esta condición, se almacena el camino indexado con k en $pensel$; en $tcod$, el primer codificador de la secuencia en el camino k; en $selsum$, el índice j correspondiente al sumidero que cumple la condición y; en $selrut$, nuevamente el camino indexado con k.

i. Si $pensel$ no está vacío, para cada camino k almacenado en éste, se actualiza la celda $tsupmat(j, fil, col)$, asignándole el residuo de la división entera por 2 de su contenido actual incrementado en 1. El índice fil es tal que $tdrmin(j).extremos(fil).pen$ es el penúltimo nodo del camino k en $tdrmin(j)$ y, el índice col es tal que $segnodtot(col)$ es el segundo nodo del camino k en $tdrmin(j)$. Si determinante de $tdsupm(j)$ es distinto de nulo, se continúa con el siguiente sumidero (se incrementa el índice j); en caso contrario, se activa la bandera swt para impedir que se continúe la revisión con las matrices asociadas a los siguientes sumideros. Si $pensel$ está vacío, se continúa la revisión con el siguiente sumidero (se incrementa el índice j).

j. Si la bandera swt permanece desactivada, se verifica que los determinantes almacenados en $tdsupm$ de las matrices asociadas a los nodos sumideros sean no nulos. Si se cumple esto, se activa la bandera sw para que no se continúe revisando la siguiente componente de $codseg$; en caso de no cumplirse, se continúa con la siguiente componente de $codseg$ (se incrementa el índice h). Si la bandera swt está activada, igualmente se continúa con la siguiente componente de $codseg$ (incremento del índice h).

Algoritmo 4.18: Verdet: Verificación de determinantes

Entrada: Arreglos $tdrmin, dest, segnodtot, nodos\_cod$ Escalar $minflujo$
Salida: Escalares $tcod, selsum, selrut, sw$

1 Inicializar $supmat(i,j,k) = 0, (\forall i = 1, \dots, |dest|, (\forall j = 1, \dots, minflujo, (\forall k = 1, \dots, |segnodtot|)))$;
2 $tcod = \emptyset$;
3 $selsum = \emptyset$;
4 $selrut = \emptyset$;
5 $supmat(i,j,k) + +, \Big(\forall i = 1, \dots, |dest|, \big(\forall j = 1, \dots, minflujo, tdrmin(i).extremos(j).listseg(l), (\forall l = 1, \dots, |tdrmin(i).extremos(j).listseg|)\big)\Big)$, $\exists k, 1 \leq k \leq minflujo$, tal que $tdrmin(i).extremos(j).listseg(l) = segnodtot(k)$;
6 $msupmat(i,j,k) = supmat(i,j,k) \bmod 2, (\forall i = 1, \dots, |dest|, \big(\forall j = 1, \dots, minflujo, (\forall k = 1, \dots, minflujo)\big))$; *//Cálculo de la matriz* $supmat$ *en el campo* $\mathbb{F}_2$
7 $dsupm(i) = \det\big(msupmat(i)\big), \forall i = 1, \dots, |dest|$; *// Almacenamiento de determinantes // en vector* $dsupm$
8 **Si** $|dsupm|_{\neq 0} == |dest|$
9 $sw = true$;
10 **Sino**
11 $sw = false$;
12 **Fin Si**
13 **Si** $\sim sw$

```
14    codseg = ∅;
15    Para i = 1:|tdrmin|
16        codseg=Conscodseg(codseg, nodos_cod, tdrmin(i));
17    Fin Para
18    h = 1;
19    Mientras ~sw ∧ h ≤ |codseg|
20        swt = false;
21        j = 1;
22        tsupmat = msupmat;
23        tdsupm = dsupm;
24        Mientras ~swt ∧ j ≤ |tdrmin|
25            pensel = ∅;
26            Para cada ruta k = 1, … , |tdrmin(j).tdrmn|
27                codint = nodos_cod ∩ tdrmin(j).tdrmn(k).camino;
28                Si codint ≠ ∅
29                    nodos = codint ‖ tdrmin(j).tdrmn(k).camino(2);
30                    Si nodos ⊆ codseg(h).nodos
31                        Adicionar la ruta k en pensel;
32                        tcod = nodos(1);
33                        selsum = j;
34                        selrut = k;
35                    Fin Si
36                Fin Si
37            Fin Para
38            Si pensel ≠ ∅  //Coincidencias con los codseg y hay penúltimos
39                Para cada ruta k en pensel
40                    tsec = tdrmin(j).tdrmn(k).camino(2);
41                    ultimo = |tdrmin(j).tdrmn(k).camino|;
42                    tpen = tdrmin(j).tdrmn(k).camino(ultimo − 1);
43                    Buscar en tdrmin(j).extremos la posición fil, tal que
                      tdrmin(j).extremos(fil).pen = tpen;
44                    Buscar en segnodtot la posición col, tal que
                      segnodtot(col) = tsec;
45                    tsupmat(j, fil, col) = (tsupmat(j, fil, col) + 1) mod 2;
46                Fin Para
47                tdsupm(j) = det(tsupmat(j)); //Actualización de determinante
                                               // del j-ésimo sumidero
48                Si tdsupm(j) ≠ 0
49                    j++; //Se continúa con el siguiente sumidero
50                Sino
51                    swt = true; //Se interrumpe la revisión de determinantes
52                Fin Si
53            Sino //No se hallaron coincidencias con codseg
54                j++;
55            Fin Si
56        Fin Mientras
```

```
57        Si ~swt
58            Si |tdsupm|_{≠0} == |dest|   //Todos los determinantes son no nulos
59                sw = true;
60            Sino
61                h++;
62            Fin Si
63        Sino
64            h++;
65        Fin Si
66    Fin Mientras
67 Fin Si
68 Retornar tcod,selsum,selrut,sw;
```

Ejemplo 4.29: Las Figuras 4.14 a 4.17 muestran el proceso de reducción de un grafo de comunicaciones G de 46 nodos, 4 sumideros, flujo máximo de 6, y 6 enlaces de salida desde el nodo fuente $s = 1$. La Figura 4.16 muestra al grafo de comunicaciones G, donde se destacan los sumideros 43, 44, 45 y 46. La Figura 4.17 muestra la primera aproximación a la solución, aplicando solo eliminación por caminos disyuntos. Se observa que los nodos 44 y 46, tienen solución, ya que logran derivar los seis paquetes enviados desde s; mientras 43 y 45, no logran la solución. La Figura 4.18 corresponde a la segunda aproximación, obtenida por el algoritmo de reducción, donde el nodo 23 deja de ser codificador al eliminar el enlace (14,23). Tampoco se logra obtener la solución para los sumideros 43 y 45; sin embargo, se mantiene la solución para los nodos 44 y 46. Por último, la Figura 4.19 muestra la solución obtenida con la aplicación del algoritmo de Verificacion de determinantes, donde el nodo 24 deja de ser codificador al eliminar el enlace (12,24). En este grafo, se logra entregar los seis paquetes a los cuatro sumideros.

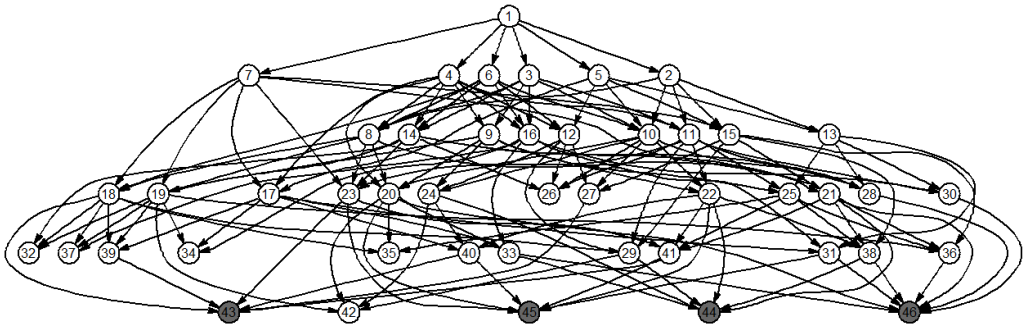


Figura 4.16 Grafo de comunicaciones G de 46 nodos, 4 sumideros, $r = 6$, $\kappa = 6$

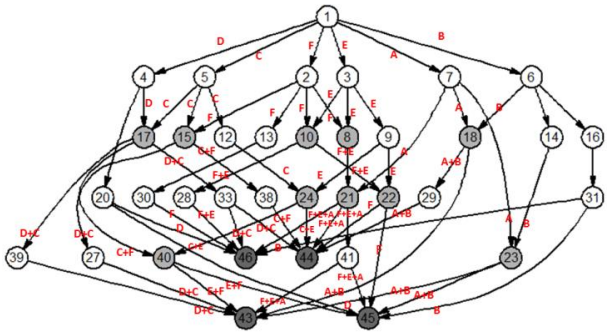


Figura 4.17 Grafo G': Aproximación inicial con caminos disyuntos. Nodos 43 y 45 sin solución

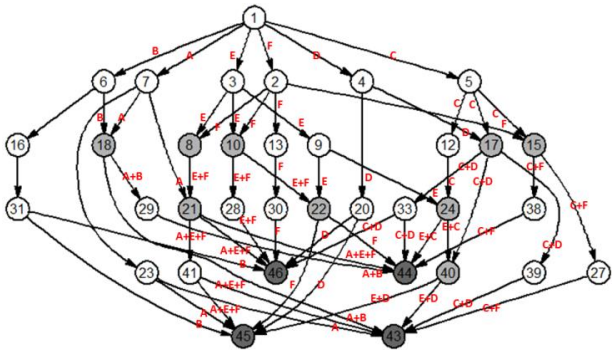


Figura 4.18 Grafo G': Segundo aproximación con reducción. Nodos 43 y 45 sin solución

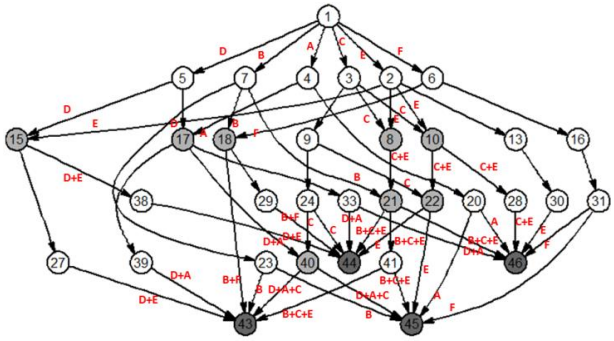


Figura 4.19 Grafo G': Tercera aproximación aplicando Verdet. Todos con solución

Capítulo 5

Modelo de Salida y Entrega de Paquetes

En este capítulo se propone un método basado en un sistema de restricciones lineales para solucionar la forma en que deben ser ordenados, en los enlaces salientes de un grafo de mínimo flujo máximo G' desde el nodo fuente s, los $f_{G'} = r$ paquetes en el espacio vectorial $\mathbb{F}_q^n$ que corresponden al flujo máximo de una sesión multicast. El ordenamiento en los enlaces de salida desde el nodo fuente s, debe garantizar la entrega correcta de los paquetes en los nodos sumideros T. Los paquetes se pueden expedir desde el nodo fuente en formato original (paquetes simples) o como combinaciones lineales de paquetes simples (paquetes combinados). A través de los enlaces y nodos interiores de la red, se pueden realizar o no, más combinaciones lineales que permitan la llegada de paquetes que conformen vectores en el espacio vectorial $\mathbb{F}_q^n$, y que sean linealmente independientes. La importancia del capítulo está en que las restricciones más relevantes que se derivan de la solución, definen cada flujo de llegada en cada sumidero como una combinación lineal de algunos de los flujos de salida en el nodo fuente en términos de los códigos que representan los paquetes. Los paquetes se etiquetan como variables basadas en un sistema de códigos dentro del espacio vectorial $\mathbb{F}_2^r$, donde las restricciones se establecen con coeficientes en $\mathbb{F}_2$.

5.1 Tasa máxima de transmisión

El interés es enviar la misma información desde el nodo fuente a cada nodo sumidero a la máxima tasa de datos posible. Sea $r(s,T)$, o simplemente r la tasa alcanzable a la cual se puede transmitir la misma información en forma confiable a cada nodo sumidero $t \in T$; se observa claramente que la tasa de transmisión o ancho de banda máxima $r(s,T)$ tiene como cota superior el mínimo de los flujos máximos entre s y cada $t \in T$, es decir $r(s,T) \leq min_{t\in T} MaxFlujo(s,t)$, o lo que es lo mismo, y dado que [112] demostró con el teorema MaxFlujo-MínCorte que el mínimo corte es igual al máximo flujo para el caso unicast, se puede establecer que $r(s,T) \leq min_{t\in T} MinCorte(s,t)$. Esta tasa de transmisión máxima solo es alcanzable en cualquier red donde se lleve a cabo una comunicación multicast utilizando Network Coding, como fue demostrado por [5].

5.2 Modelo de codificación de mensajes

Sea $c(p)$ el código del paquete p, los r paquetes simples y los combinados que se constituyan a lo largo de la transmisión hacia los nodos en T se representarán (o etiquetarán) por códigos que pertenecen al conjunto

$$\wp = \{c(p) | c(p) \in [1: 2^r - 1]\} \tag{5.1}$$

Los códigos de paquetes simples constituyen el conjunto $\wp_u$, tal que $\wp_u \subset \wp$, donde

$$\wp_u = \{c(p)|c(p) = u_i = [0^{i-1}, 1, 0^{r-i}], i = 1, \dots, r\} \quad (5.2)$$

El vector u_i es el i-$ésimo$ vector de la base canónica de $\mathbb{F}_2^r$. El número de códigos en $\wp$ que se puede manejar en una sesión multicast de flujo máximo r es $|\wp| = 2^r - 1$ y el número de códigos correspondientes a paquetes simples es $|\wp_u| = r$.

Además, los códigos (o etiquetas) de los paquetes simples también se representan con mnemónicos correspondientes a un alfabeto $\Sigma = \{A_1, A_2, \dots, A_r\}$ y se relacionan de acuerdo con el ordenamiento topológico de los símbolos del alfabeto, según se especifica en la siguiente función biyectiva:

$$g: \wp_u \mapsto \Sigma \quad (5.3)$$
$$g(u_i) = A_i$$

De la función g, se deriva que $\Sigma = \{A_i|A_i = g(u_i),\ u_i \in \wp_u\}$, y $|\Sigma| = r$.

Ejemplo 5.1: Si $r = 2$, los conjuntos $\wp, \wp_u$ y Σ son:
$\wp = \{01,10,11\}$, $\wp_u = \{01,10\}$ y $\Sigma = \{A, B\}$.
Donde $g(01) = A$ y $g(10) = B$ y los códigos en $\wp$ se representan como las combinaciones lineales de los símbolos en Σ: $A, B, A + B$, respectivamente.

Si $r = 3$, los conjuntos $\wp, \wp_u$ y Σ son:
$\wp = \{001,010,011,100,101,110,111\}$, $\wp_u = \{001,010,100\}$ y $\Sigma = \{A, B, C\}$
Donde $g(001) = A$, $g(010) = B$ y $g(100) = C$ y los códigos en $\wp$ se representan como las combinaciones lineales de los símbolos en Σ: $A, B, A + B, C, A + C, B + C, A + B + C$, respectivamente.

De lo anterior, se deduce que los códigos que representan a los r paquetes que se pueden transmitir desde el nodo fuente s en una sesión multicast, son elementos no nulos del espacio vectorial $\mathbb{F}_2^r$. Es decir, la diferencia entre los paquetes a transmitir está dada por los códigos que los representan y no por sus contenidos. Igualmente, si los nombres o códigos para identificar dos o más paquetes son diferentes, también lo son en sus contenidos.

En cada nodo de codificación/enrutamiento se establece un paquete de salida que corresponde a una combinación lineal de los paquetes entrantes, utilizando como coeficientes de codificación global [18] los símbolos (bits) con valor en 1 que pertenecen al campo $\mathbb{F}_2$. Esta combinación lineal se traduce en una suma *bit a bit* entre los bits componentes de los paquetes que ingresen al nodo.

Por ejemplo, si se observa la Figura 5.1, cada paquete p que llega al nodo de codificación c, tiene longitud uno, es decir $p \in \mathbb{F}_2^l$ y, cada $a_i, b_i \in \mathbb{F}_2$. En el nodo c se establece la combinación lineal que emerge por cada enlace de salida. El nuevo paquete, también de longitud l, se reenvía *bit a bit* durante el proceso de transmisión y propagación, desde el nodo c hacia cada uno de los nodos adyacentes conectados a través de los enlaces salientes.

Es importante resaltar que, como consecuencia de este planteamiento, se llevan a cabo l sumas *bit a bit* entre ambos paquetes para determinar el paquete combinado.

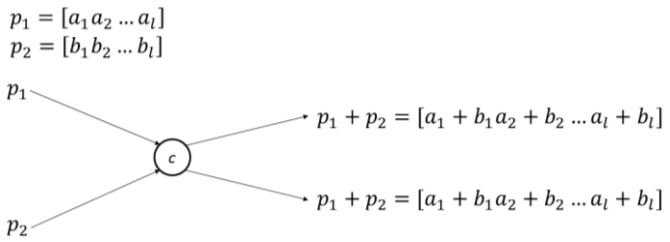


Figura 5.1 Combinación lineal bit por bit en paquetes

5.3 Problema Propuesto

5.3.1 Solución al Sistema Lineal de Ecuaciones sobre Network Coding para Multicast

La solución de un sistema lineal de ecuaciones sobre Network Coding para una sesión multicast, ha sido abordado de distintas formas según lo explicado en la sección 2.4.4, pero en todos los casos se ha revisado desde el contenido de los mensajes o paquetes, y esto sujeta la solución al tamaño del campo al que pertenece el contenido.

Una sesión multicast para una red de comunicaciones, se configura con un nodo fuente s, un conjunto de nodos sumideros T, una capacidad unitaria de información por enlace de longitud $l\ bits$ y un flujo máximo r para cada nodo en T, tal como lo muestra la Figura 5.2.

La meta es determinar la etiqueta o código de paquete en $\wp$ para cada uno de los κ enlaces de salida $e_1, e_2, \dots, e_\kappa$ que emergen del nodo fuente s. Este proceso garantiza la recepción de las combinaciones lineales independientes en los nodos sumideros para recuperar los códigos en $\wp_u$ de los r-paquetes simples originales de longitud $l\ bits$. Los paquetes son enviados a través de los enlaces y nodos interiores de G', y etiquetados dentro del conjunto $\wp$ o especificados como combinaciones lineales de los mnemónicos en Σ.

La meta consiste en acomodar en cada enlace de salida $e_1, e_2, \dots, e_\kappa$ a partir de s, la mejor combinación de etiquetas (vectores u_i o sus combinaciones lineales) en $\wp$ para atravesar la red internamente (nodos de enrutamiento y codificadores) y obtener las etiquetas de los paquetes originales en los nodos de T a través de la resolución de un sistema de ecuaciones linealmente independientes. Es decir, las combinaciones lineales que se generen en los r caminos que alcancen cada nodo en T desde s, deben ser linealmente independientes.

La solución se establece sin tener en cuenta el tamaño del campo al que pertenecen los contenidos de los paquetes; es decir, independientemente de su longitud en bits. Solo se tendrá en cuenta el tamaño de la etiqueta o código de cada paquete que se genere en el nodo fuente.

En la Figura 5.2, se observa que por cada enlace de salida, a partir de s, puede emerger uno de los paquetes que se generen con etiqueta $A_i = g(u_i)$ o combinaciones lineales de los códigos que representen las etiquetas en $\wp_{\mathrm{u}}$. A cada nodo sumidero debe entrar un flujo de paquetes, donde los códigos que los representan están en $\wp$ y deben ser linealmente independientes. Los códigos entrantes son el resultado de las combinaciones lineales a lo largo de la transmisión a través de los nodos interiores.

5.3.2 Cota superior del número de salidas

Es importante resaltar que la solución no es única para cualquier grafo $G' = (V', E')$ que represente una sesión multicast. Se denota por σ el número de posibles combinaciones de paquetes enviados a través de los enlaces salientes de s. Las soluciones se podrían plantear en dos términos: para paquetes simples y para paquetes combinados:

El número máximo de combinaciones para paquetes simples emergentes desde el nodo s a través de sus enlaces de salida es:

$$\sigma = r^{\kappa} \tag{5.4}$$

Si se considera que el nodo (enrutador) s puede generar combinaciones lineales de los paquetes simples entrantes a él, se podrían obtener hasta un límite máximo de combinaciones determinadas por:

$$\sigma = \left(\sum_{i=1}^{r} \binom{r}{i}\right)^{\kappa} = (2^r - 1)^{\kappa} \tag{5.5}$$

Es importante especificar que todas las combinaciones no son válidas para resolver el problema multicast planteado, ya que no tendrían solución en los enlaces de llegada a los nodos sumideros.

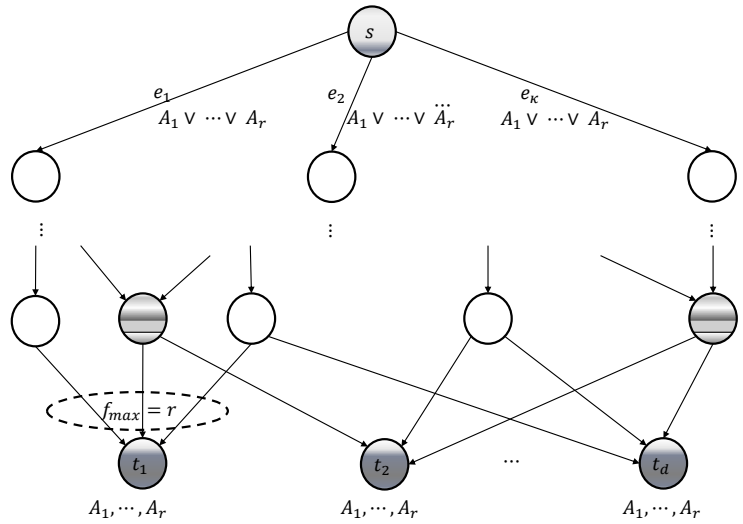


Figura 5.2 Visión general del problema

5.4 Representación del Modelo de Red

5.4.1 Matriz de Adyacencia de la red

La red de comunicaciones para una sesión multicast, con las diferentes rutas desde el nodo fuente s hasta los distintos nodos sumideros en $T = \{t_1, t_2, \ldots, t_d\}$, se representa con una matriz de adyacencia. Además, cada celda en la matriz representa la capacidad de cada uno de los enlaces que conforman la red después de haber obtenido los caminos disyuntos desde el nodo fuente s hacia cada uno de los nodos sumideros. Esta matriz de capacidades se denomina C y es de tamaño $|V'| \times |V'|$, donde:

$$C(i,j) = \begin{cases} 0, & (i,j) \notin E' \\ 1, & (i,j) \in E' \end{cases} \tag{5.6}$$

Como se especificó previamente, en el enlace $e = (i,j), i,j \in V', i < j$, de esta forma, también se logra un ordenamiento topológico de los enlaces en E'.

5.4.2 Clasificación de enlaces

Un enlace $e \in E'$, se puede clasificar, según sí $o(e)$ es el nodo fuente o no, y según si $d(e)$ está en T o no, en cualquiera de los siguientes tipos:

$$Tipo(e) = \begin{cases} 0, & o(e) \neq s, d(e) \notin T \\ 1, & o(e) = s, d(e) \notin T \\ 2, & o(e) \neq s, d(e) \in T \\ 3, & o(e) = s, d(e) \in T \end{cases} \tag{5.7}$$

Los Tipo 0 corresponden a los *enlaces interiores*, los cuales originan en un nodo distinto del fuente s, y sus nodos destinos no pertenecen al conjunto T. Los Tipo 1 serán llamados *enlaces de salida*, ya que originan en el nodo fuente s y, finalizan en un nodo cualquiera distinto de los que están en T. Los Tipo 2 se denominarán *enlaces de llegada*, ya que originan en un nodo distinto del fuente s y finalizan en un nodo perteneciente al conjunto T. Finalmente, los Tipo 3 se denominarán enlaces *salida-llegada*, ya que pueden enviar paquetes directamente desde el nodo fuente s hasta un sumidero en T, sin pasar por *enlaces interiores*.

Es claro que a cada nodo sumidero llega un flujo de paquetes cuya cantidad se constituye en la cota límite de la red multicast y, dado que la capacidad de cada enlace $e \in E'$, $C(e) = 1$, el número de *enlaces de llegada* que llevan el flujo a cada nodo sumidero es igual a:

$$r(s,T) \leq min_{t \in T} MinCorte(s,t) = min_{t \in T} MaxFlujo(s,t) \tag{5.8}$$

Se denota por S el conjunto de *enlaces de salida*, que corresponde con la siguiente definición:

$$S = \{e \in E' \mid o(e) = s, \ d(e) \notin T\} \tag{5.9}$$

El conjunto de *enlaces de llegada* se denota por L y corresponde con la siguiente definición:

$$L = \{e \in E' \mid o(e) \neq s, d(e) \in T\} \tag{5.10}$$

El conjunto de *enlaces de salida-llegada* se denota por K y corresponde con la siguiente definición:

$$K = \{e \in E' \mid o(e) = s, d(e) \in T\} \tag{5.11}$$

De la sección 5.3.1, y las ecuaciones (5.9) y (5.11), el número de enlaces emergentes de s es $|S \cup K| = |S| + |K| = \kappa$; y de (5.10) y (5.11) se establece que $|L \cup K| = |L| + |K| = r|T|$.

5.4.3 Tabla de enlaces

A partir de la matriz de adyacencia C, la cual representa a G', y la clasificación de los enlaces, se construye la tabla de *Enlaces*, la cual tiene la siguiente estructura (Tabla 5.1).

Tabla 5.1 Formato Tabla de Enlaces

Índice	l
Origen	i
Destino	j
Tipo	0,1,2,3

Si $C(i,j)=1$, el enlace e_l con índice l contiene $Origen(l)=i$ (es lo mismo que $o(l)=i$) y $Destino(l)=j$ (es lo mismo que $d(l)=j$). El $Tipo(l)$ está especificado según lo definido previamente. En (5.12) se especifica el número de enlaces que existen en G'.

$$\lambda=\sum_{i=1}^{|V'|}\sum_{j=1}^{|V'|}C(i,j) \tag{5.12}$$

5.5 Bases de la Solución

5.5.1 Grafo lineal dirigido etiquetado y Matriz de un salto

El Grafo Lineal Dirigido Etiquetado o *Directed Labeled Line Graph* (DLLG) [17], [82] derivado de $G'=(V',E')$ es $\mathfrak{G}=(\mathcal{V},\mathcal{E})$, el cual se construye a partir de la matriz de capacidades C con las siguientes reglas:

1. Cada nodo $v\in\mathcal{V}$ corresponde con un enlace dirigido $e=(i,j)\in E'$. Luego $\mathcal{V}=E'$.
2. Cada enlace $e'\in\mathcal{E}$ es definido como $e'=(e_1,e_2)$, donde $e_1,e_2\in E'$ y $d(e_1)=o(e_2)$, por tanto $\mathcal{E}=\{(e_1,e_2)\in E'\times E'|d(e_1)=o(e_2)\}$.
3. Se construye la matriz de adyacencia $F=\left(F(k,l)\right)_{k=1,l=1}^{|\mathcal{V}|}$ para el grafo $\mathfrak{G}$, donde cualquier enlace $e'=(e_1,e_2)\in\mathcal{E}$ se etiqueta según (5.13):

$$F(e_1,e_2)=\begin{cases}1, & d(e_1)=o(e_2)\\ 0, & \text{en otro caso}\end{cases} \tag{5.13}$$

La matriz de un salto F, resultante de las tres reglas anteriores, tiene las siguientes características:

1. Es una matriz diagonal superior con ceros en la diagonal principal porque se construye a partir de un grafo dirigido y acíclico, cuyos nodos y enlaces están etiquetados en orden topológico.
2. Por la razón anterior, es una matriz nilpotente (Lema 2.2), es decir $\exists n\in\mathbb{N}$, tal que $F^n=0$.
3. Dada la propiedad de nilpotencia de F, la matriz $F^h=\left(F^h(k,l)\right)_{k=1,l=1}^{|\mathcal{V}|}$ es generada, donde cada $F^h(k,l)$, $1\le h\le n$ se interpreta según su valor como:

$$F^h(k,l)=\begin{cases}i, & \text{existen } i \text{ caminos desde el enlace } k \text{ hasta el } l \text{ en } h \text{ saltos}\\ 0, & \text{no existen caminos desde el enlace } k \text{ hasta el } l \text{ en } h \text{ saltos}\end{cases} \tag{5.14}$$

5.5.2 Matriz de saltos Fuente-Sumideros

A partir de la matriz de un salto y sus propiedades, y por medio del Algoritmo 5.1, se construye la matriz de saltos Fuente-Sumideros Q, la cual tiene una dimensión de $(|S|+|K|)\times|E'|$.

Algoritmo 5.1: Construcción_Matriz_Saltos_Fuente-Sumideros

Entrada: Arreglos $F, S \cup K, E'$
Salida: Arreglo Q

```
//Las  filas de Q están constituidas por los enlaces k, tal que k ∈ S ∪ K.
//Las  columnas  de  Q  están  constituidas  por  todos  los  enlaces  l ∈ E'
//(incluyendo los enlaces interiores o tipo 0, que no son necesarios en
//el modelo final).
1  Inicializar Q, con Q(k,l) = 0, (∀k ∈ S ∪ K, (∀l ∈ E'));
2  Para cada F^h, (1 ≤ h < n) //Cuando h = n, F^n = 0
3     Para cada k ∈ S ∪ K
4        Para cada l ∈ E'
5           Si F^h(k,l) ≥ 1 y Tipo(l) = 2
6              Q(k,l)+= F^h(k,l); // Incrementa en F^h(k,l) cada vez
7           Fin Si
8        Fin Para
9     Fin Para
10 Fin Para
11 Retornar Q;
```

En conclusión, cada $Q(k,l) \neq 0$, representa la cantidad de veces que el flujo proveniente del enlace de salida k ($Tipo(k) = 1$) contribuye, en una combinación lineal, al cálculo del flujo del enlace de llegada l ($Tipo(l) = 2$). Es decir, si f_k corresponde al flujo saliente de s y transmitido a través del enlace de salida k, luego $Q(k,l) * f_k$ será el flujo que transitará en una combinación lineal a través del enlace de llegada l hasta el nodo sumidero $d(l)$. Por tanto, f_k contribuirá $Q(k,l)$ veces a la combinación lineal del flujo entrante a través del enlace de llegada l hacia el nodo sumidero $d(l)$. Realmente, el flujo con el que contribuirá el enlace de salida k, será f_k si $Q(k,l)$ es un número entero impar; y será cero, en caso contrario.

Definición 5.1: Sea $\mathcal{L} = \{l \in E' \mid d(l) \notin T\}$, cuyos destinos no están en T. Es decir, $Tipo(l) = 0$ o $Tipo(l) = 1$.

A partir de esta definición y definiciones anteriores, se pueden establecer los siguientes teoremas:

Teorema 5.1: Sea $l \in L$, luego $Q^T(l) \neq 0$.

Demostración:
Si $l \in L$, por definición del conjunto L, $d(l) \in T$. Además, $Tipo(l) = 2$, por tanto en cada iteración del Algoritmo 5.1 hasta llegar a $F^n = 0$, se cumple que cuando haya h saltos de k a l; es decir $F^h(k,l) = i$, se logrará el incremento en la celda $Q(k,l)$ y esto evitará que $Q^T(l) = 0$. ■

Teorema 5.2: Sea $l \in \mathcal{L}$, luego $Q^T(l) = 0$.

Demostración:
Si $l \in \mathcal{L}$, por definición del conjunto $\mathcal{L}$, $d(l) \notin T$. Además, $Tipo(l) \neq 2$ y $Tipo(l) \neq 3$, por tanto en cada iteración del Algoritmo 5.1 hasta llegar a $F^n = 0$, no se logrará el incremento en la celda $Q(k,l)$ y esto conllevará que $Q^T(l) = 0$. ■

Teorema 5.3: Si l es un enlace donde $Tipo(l) = 3$**,** luego $Q^T(l) = 0$**.**

Demostración:
Si l es un enlace Tipo 3, luego $l \in K$ y, por definición, $o(l) = s$ y $d(l) = t \in T$. Esto significa que l es un enlace directo desde s a t sin nodos intermediarios, por tanto no hay enlaces que precedan o sucedan a l y, consecuentemente, l es un enlace de llegada sin saltos, llevando a que en cada iteración del Algoritmo 5.1, $F^h(k,l) = 0, \forall k \in S \cup K$ y resultando al final en $Q^T(l) = 0$. ■

Q contiene $|\mathcal{L}|$ columnas en 0 que se denominarán *columnas no sumideros*, las cuales no muestran ninguna relación entre algún enlace de salida y un enlace de llegada, o lo que es lo mismo, entre el nodo s y algún $t \in T$. Estas cumplen con el Teorema 5.2.

Al resto, se les denominará *columnas sumideros*, las cuales están constituidas por las columnas l, cuyo $d(l) \in T$. Es decir, las columnas que cumplen con el Teorema 5.1 o Teorema 5.3.

El tiempo de ejecución del Algoritmo 5.1 es $O(n \times |S \cup K| \times |E'|) = O(n\kappa|E'|)$.

5.5.3 Reducción de la matriz de saltos Fuente-Sumideros

En la matriz de saltos Fuente-Sumideros Reducida, se eliminan las $|\mathcal{L}|$ columnas en 0 que corresponden con enlaces $l' \in E'$, tal que $d(l') \notin T$; es decir, solo se mantienen las columnas que corresponden a enlaces tipo 2 ó 3. Se denota por Q_r la matriz de saltos Fuente-Sumideros Reducida y su dimensión está dada por $(|S| + |K|) \times (|L| + |K|) = \kappa r|T|$. El Algoritmo 5.2 muestra el proceso de reducción de la matriz Fuente-Sumideros.

Algoritmo 5.2: Reducción_Matriz_Fuente-Sumideros

Entrada: Arreglos $Q, L \cup K, S \cup K$
Salida: Arreglo Q_r

```
1  Inicializar Q_r, con Q_r(k,l) = 0, (∀k ∈ S ∪ K, (∀l ∈ L ∪ K))
2  Para cada enlace l ∈ L ∪ K
3      Q_r^T(l) = Q^T(l);
4  Fin Para
5  Retornar Q_r;
```

El tiempo de ejecución del Algoritmo 5.2 es $O(|S \cup K| \times |L \cup K|) = O(\kappa r|T|)$.

5.6 Construcción de restricciones

5.6.1 Principios del método planteado

El modelo propuesto parte del grafo $G' = (V', E')$ representando una red multicast con un nodo fuente s, un conjunto de nodos sumideros T, con flujo máximo r y capacidad $C(e) = 1$ para cada enlace $e \in E'$. El objetivo es construir un sistema de ecuaciones lineales formadas por variables que representan a los flujos de paquetes de los *enlaces de salida*, *llegada y salida-llegada*, donde cada flujo está constituido por el código asociado al paquete que puede viajar por este (puede ser un paquete simple o combinado). El modelo se basa en tres principios:

5.6.1.1 Definición de restricciones en el nodo fuente s

En el nodo fuente s se originan r paquetes distintos, distinguidos con una etiqueta de longitud r-*bits* y de la forma $A_i = g(u_i)$, donde u_i es el vector de módulo unitario, definido en la sección 5.2. De este nodo emergen $|S| + |K|$ enlaces (entre *salida* y *salida-llegada*), permitiendo establecer las r posibles salidas de paquetes por cada uno de estos enlaces. Se deduce que se pueden obtener los siguientes flujos simples (un paquete simple por enlace de salida o de salida-llegada) según la relación (5.15):

$$f_e = A_1 | A_2 | \dots | A_r, \forall e \in S \cup K \tag{5.15}$$

Es posible que por cada enlace de salida $e \in S \cup K$ emerjan combinaciones lineales (paquetes combinados) de paquetes simples etiquetados con algún u_i, generando un paquete p_e con etiqueta $c(p_e) \in \wp$ que se almacena en el flujo combinado f_e:

$$f_e = \sum_{u_i \in \wp} u_i = \sum_{u_i \in \wp, g(u_i) = A_i} A_i \,, \forall e \in S \tag{5.16}$$

La ecuación anterior también se puede escribir en la forma:

$$f_e \neq 0 \tag{5.17}$$

Para todos los casos $f_e \in \mathbb{F}_2^{\mathrm{r}} \setminus \{0\}$ o $f_e \in \wp$.

5.6.1.2 Definición de restricciones basadas en los saltos desde un enlace de salida a uno de llegada
Dada la gráfica de la Figura 5.3, sea f_{sv_1} un flujo que emerge desde s, el cual llegará a t a través de los distintos enlaces que forman el camino y, por el principio de conservación del flujo (3.1.1), se desprende la siguiente la relación de igualdad (5.18):

$$f_{sv_1} = f_{v_1 v_2} = \cdots = f_{v_{j-1} v_j} = f_{v_j t} \tag{5.18}$$

Figura 5.3 Camino del nodo fuente a un nodo sumidero

Esta igualdad, no se debe tomar en un sentido estrictamente literal. En un camino desde el nodo fuente s hasta un nodo sumidero t, sean f_k y f_l los flujos a través de los enlaces k y l, tal que $f_k = f_l$, donde $d(k) = o(l)$. Se interpreta como que el flujo a la izquierda de la igualdad contribuye al flujo de la derecha. Es decir, hará parte de la combinación lineal que se formará para obtener el flujo a través de cada enlace saliente en cada nodo intermedio. Por tanto, el flujo de un enlace saliente se ve incrementado por el flujo que le precede en la igualdad.

De esta igualdad se pueden desprender dos escenarios:

a. Si $d(k)$ es un nodo de enrutamiento clásico, el flujo entrante f_k es el mismo flujo saliente f_l, ya que no existe codificación lineal en la salida y simplemente el flujo pasa del enlace entrante al enlace saliente en forma idéntica.
b. Si $d(k)$ es un nodo codificador, el flujo entrante f_k es una componente de la combinación lineal formada en los enlaces salientes del nodo. Es decir,

$$f_l = \cdots + f_k + \cdots \tag{5.19}$$

Del escenario en (a) y de (5.18), si no existe ningún nodo de codificación en el camino desde s hasta t, se puede deducir que:

$$f_{sv_1} = f_{v_j t} \tag{5.20}$$

Es decir, el flujo que se originó en el nodo fuente s, y que se transmite a través del enlace de salida (s, v_1), se mantiene después de j saltos en el enlace de llegada (v_j, t).

Si existe un nodo codificador en el grafo G', se generarán rutas desde el nodo fuente s hasta un nodo sumidero t, como muestra la Figura 5.4, que corresponde a dos rutas emergentes desde s que convergen en el nodo codificador v_{j-1}.

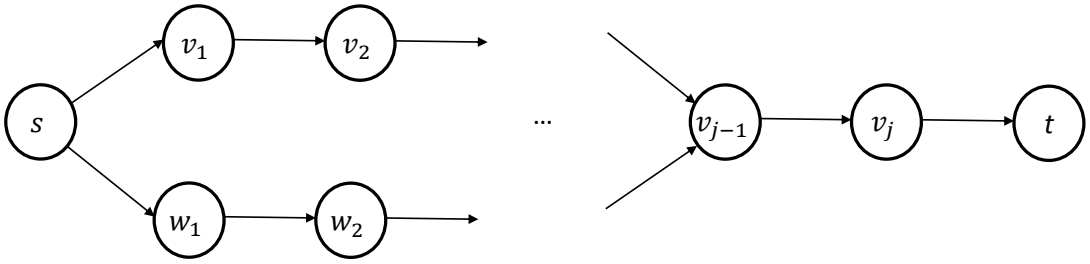


Figura 5.4 Confluencia de rutas en nodo codificador

Por tanto, de aquí se desprenden las dos siguientes relaciones de igualdad de flujos desde s hasta t:

$$f_{sv_1} = f_{v_1 v_2} = \cdots = f_{v_{j-1} v_j} = f_{v_j t} \tag{5.21}$$

y

$$f_{sw_1} = f_{w_1 w_2} = \cdots = f_{v_{j-1} v_j} = f_{v_j t} \tag{5.22}$$

Se observa que (v_{j-1}, v_j) es un enlace cuello de botella, y el flujo $f_{v_{j-1} v_j}$ nace en el nodo codificador v_{j-1}. Además, los flujos f_{sv_1} y f_{sw_1} emergen de s a través de los enlaces (s, v_1) y (s, w_1), respectivamente. Según (5.21) y (5.22), estos contribuyen al flujo $f_{v_{j-1} v_j}$ en el enlace cuello de botella, y por (5.19) forman la combinación lineal:

$$f_{v_{j-1} v_j} = f_{sv_1} + f_{sw_1} \tag{5.23}$$

En consecuencia, de (5.21),(5.22) y (5.23), se puede deducir que el flujo en el enlace de llegada (v_j, t) dependerá directamente de los flujos salientes de s a través de los enlaces de salida (s, v_1) y (s, w_1) mediante la relación:

$$f_{v_j t} = f_{sv_1} + f_{sw_1} \tag{5.24}$$

Tanto en (5.21) como en (5.24), los flujos en los enlaces de llegada al nodo $t \in T$ corresponden a paquetes de tamaño l-bits en el espacio vectorial $\mathbb{F}_2^l$ como se especificó en la sección 2.5.5, al igual que los paquetes de los flujos salientes.

5.6.1.3 Definición de restricciones en los enlaces de llegada

En cada nodo sumidero $t \in T$ se deben cumplir las siguientes reglas:

a. Flujos de llegada deben ser no nulos: Si $f_{v_j t}$ es el flujo que arriba a t, a través del enlace de llegada (v_j, t), el paquete que fluye a través de éste y entrante a t, debe tener una etiqueta o combinación lineal con código distinto de 0, es decir, según la Figura 5.5, se debe cumplir que:

$$f_{v_j t} \neq 0, \forall (v_j, t) \in L \tag{5.25}$$

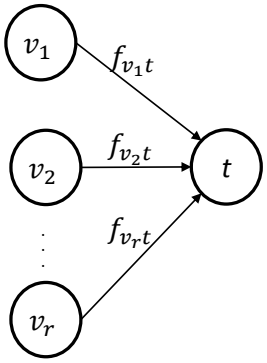


Figura 5.5 Enlaces de llegada

b. La combinación lineal de los flujos de llegada a un nodo sumidero debe ser no nula: De igual forma se debe cumplir que la combinación lineal de los códigos de los paquetes que ingresan a cada nodo sumidero, a través de los flujos de llegada, debe ser distinta de 0. Es decir, según la Figura 5.5 se debe cumplir la restricción (5.26).

$$\sum_{j=1}^{r} f_{v_j t} \neq 0, \forall t \in T \tag{5.26}$$

c. La matriz de códigos que etiquetan a los flujos de enlaces de llegada, debe tener rango r: Los códigos de paquetes que llegan dentro de cada flujo al nodo t, deben ser, en conjunto, linealmente independientes, es decir los *bits* que conforman el código de cada paquete en cada flujo entrante a un nodo sumidero t, forman las filas de una matriz cuadrada P de dimensión $r \times r$ con $\text{rango}(P) = r$ ó $\det(P) \neq 0$. Las filas de P representan los códigos de los r-paquetes de tamaño r-bits que ingresan al nodo sumidero.

La forma de la matriz P, constituida por los r-códigos que identifican los paquetes de los flujos que arriban a un nodo sumidero en T, es:

$$P = \begin{bmatrix} c(p_1) \\ c(p_2) \\ \vdots \\ c(p_r) \end{bmatrix} = \begin{bmatrix} b_{11} & b_{12} & \dots & b_{1r} \\ b_{12} & b_{22} & \dots & b_{2r} \\ \vdots & \vdots & \ddots & \vdots \\ b_{r1} & b_{r2} & \dots & b_{rr} \end{bmatrix} \tag{5.27}$$

Ejemplo 5.2: Si $r = 3$, significaría un flujo máximo para una sesión multicast de 3 paquetes generados en el nodo fuente y llegando a cada nodo sumidero. La longitud de los paquetes es indiferente. Por lo anterior, los códigos están en el espacio vectorial $\mathbb{F}_2^3$. Si se supone que a un nodo sumidero le llegan, por sus tres enlaces, los paquetes etiquetados con mnemónicos $A, A + C$ y $B + C$, estos corresponden con los siguientes códigos binarios:

$$A = 001 = P1$$
$$A + C = 101 = P2$$
$$B + C = 110 = P3$$

La matriz P correspondiente es:

$$P = \begin{bmatrix} 0 & 0 & 1 \\ 1 & 0 & 1 \\ 1 & 1 & 0 \end{bmatrix}$$

y su determinante es 1.

5.6.2 Construcción del sistema de restricciones

Las restricciones se construyen con base en las siguientes reglas:

a. Restricciones en los flujos de enlaces de salida: Para cada enlace e de tipo 1 ó 3, se definen los nombres de variables de los flujos (paquetes) de los enlaces de salida de acuerdo con la siguiente regla: Si e es un enlace de salida en $S \cup K$, su $o(e) = s$, luego la notación de la variable de flujo del enlace de salida es $f_{s\_d(e)}$ y su tamaño computacional es de r-bits.

   Las alternativas de solución están determinadas por flujos de salida según:

   1. Paquetes simples: Cada $f_{s\_d(e)}$ corresponde a un entero potencia de 2, desde 1 hasta 2^r:

$$f_{s\_d(e)} = 1 \vee 2 \vee 4 \vee \ldots \vee 2^r \tag{5.28}$$

   2. Paquetes combinados: Cada $f_{s\_d(e)}$ corresponda con cualquier entero entre 1 y 2^r:

$$f_{s\_d(e)} = [1 : 2^r] \neq 0 \tag{5.29}$$

b. Restricciones en los flujos de enlaces de llegada: Para cada enlace l de tipo 2 ó 3, se definen los nombres de variables de los flujos (paquetes) de los enlaces de llegada de acuerdo con la siguiente regla: Si l es un enlace de llegada en $L \cup K$, su $d(l) = t, t \in T$, luego la notación de la variable de flujo del enlace de llegada es $f_{o(l)\_t}$ y su tamaño computacional es de r-bits.

   Las restricciones se establecen con el apoyo de la matriz de saltos Fuente-Sumideros reducida Q_r, donde cada $Q_r(k, l) = j$ significa que existe un camino directo, con j saltos ($1 \leq j \leq n - 1$, donde $F^n = 0$) entre el enlace de salida k y el enlace de llegada l. Se observa que $Q_r^T(l)$ es la columna correspondiente al enlace de llegada l de la matriz Q_r, y $Q_r^T(l, k)$ es el número de veces que el flujo proveniente del enlace de salida k (el cual se denota $f_{s\_d(k)}$) contribuirá dentro de la combinación lineal que se constituirá para dar solución al flujo del enlace de llegada l (el cual se denota $f_{o(l)\_t}$). Para un enlace de llegada l, el flujo $f_{o(l)\_t}$ se forma como una combinación lineal constituida por la suma XOR del siguiente número de componentes:

$$\eta_{f_{o(l)\_t}} = \sum_{k \in S \cup K} Q_r^T(l, k) \tag{5.30}$$

   El flujo $f_{o(l)\_t}$ en el enlace de llegada l, se obtiene a través de la siguiente expresión, compuesta por la suma XOR de los flujos de salida $f_{s\_d(k)}, \forall k \in S \cup K$:

$$f_{o(l)\_t} = \sum_{k \in S \cup K} \sum_{i=1}^{Q_r^T(l,k)} f_{s\_d(k)} \tag{5.31}$$

c. Restricciones en los enlaces de llegada y en los nodos sumideros: Se deben cumplir las restricciones dadas en la sección 5.6.1.3.

1. Para cada enlace l de tipo 2 o 3, teniendo en cuenta (5.25), su flujo debe cumplir con la siguiente expresión:

$$f_{o(l)\_t} \neq 0, \forall t \in T, \forall l \text{ con } d(l) = t \tag{5.32}$$

2. Para la combinación lineal de los flujos en los enlaces de llegada $l_1, l_2, \ldots, l_r$ al nodo sumidero t y teniendo en cuenta (5.26), se debe cumplir con la siguiente expresión:

$$\sum_{i=1}^{r} f_{o(l_i)\_t} \neq 0, \forall t \in T, \forall l_i \text{ con } d(l_i) = t \tag{5.33}$$

3. Para establecer la independencia lineal entre los códigos de los flujos (paquetes) que arriban a un nodo sumidero $t \in T$, se construye una matriz P_t, cuyas filas contienen estos códigos. Cada uno de los r-bits que componen un flujo $f_{o(l_i)\_t}$ será denotado por $f_{o(l_i)\_t}^{(j)}$, donde $1 \leq i, j \leq r$.

$$P_t = \underbrace{\begin{bmatrix} f_{o(l_1)\_t} \\ f_{o(l_2)\_t} \\ \vdots \\ f_{o(l_r)\_t} \end{bmatrix}}_{r-bits} = \begin{bmatrix} f_{o(l_1)\_t}^{(1)} & f_{o(l_1)\_t}^{(2)} & \cdots & f_{o(l_1)\_t}^{(r)} \\ f_{o(l_2)\_t}^{(1)} & f_{o(l_2)\_t}^{(2)} & \cdots & f_{o(l_2)\_t}^{(r)} \\ \vdots & \vdots & \ddots & \vdots \\ f_{o(l_r)\_t}^{(1)} & f_{o(l_r)\_t}^{(2)} & \cdots & f_{o(l_r)\_t}^{(r)} \end{bmatrix} \tag{5.34}$$

En este trabajo, el cálculo del determinante se realizará por el método de menores y cofactores en forma recursiva sobre la matriz P_t; sin embargo, se puede llevar a cabo por cualquier otro método. Este determinante debe ser no nulo para cumplir la condición de independencia lineal. El determinante se calculará de la siguiente forma para el nodo sumidero t:

$$\det(P_t) = f_{o(l_1)\_t}^{(1)} * M_{11} + f_{o(l_1)\_t}^{(2)} * M_{12} + \cdots + f_{o(l_1)\_t}^{(r)} * M_{1r} \neq 0 \tag{5.35}$$

O lo que es lo mismo:

$$\det(P_t) = \sum_{i=1}^{r} f_{o(l_1)\_t}^{(i)} * M_{1r} \neq 0, \forall t \in T \tag{5.36}$$

En síntesis, se tiene un sistema de restricciones, que de tener solución, lograría determinar cuál es el valor del código que etiqueta al paquete que debe emerger por cada enlace de salida que nace en el nodo fuente s. De igual forma, se obtendría la solución de los valores de los códigos que arribarían a cada nodo sumidero en T. El sistema de restricciones para encontrar la solución a la sesión multicast de un nodo fuente único s y el conjunto T de nodos sumideros se resume en la Figura 5.6:

$$f_{s\_d(k)} = 1 \vee 2 \vee 4 \vee \ldots \vee 2^r \quad \text{ó} \quad f_{s\_d(k)} = [1{:}\,2^r] \neq 0$$

$$f_{o(l)\_t} = \sum_{k \in S \cup K} \sum_{i=1}^{Q_r^T(l,k)} f_{s\_d(k)}$$

$$f_{o(l)\_t} \neq 0, \forall t \in T, \forall l \text{ con } d(l) = t$$

$$\sum_{i=1}^{r} f_{o(l_i)\_t} \neq 0\,, \forall t \in T, \forall l_1 \text{ con } d(l_i) = t$$

$$P_t = \underbrace{\begin{bmatrix} f_{o(l_1)\_t} \\ f_{o(l_2)\_t} \\ \vdots \\ f_{o(l_r)\_t} \end{bmatrix}}_{r-bits} = \begin{bmatrix} f_{o(l_1)\_t}^{(1)} & f_{o(l_1)\_t}^{(2)} & \cdots & f_{o(l_1)\_t}^{(r)} \\ f_{o(l_2)\_t}^{(1)} & f_{o(l_2)\_t}^{(2)} & \cdots & f_{o(l_2)\_t}^{(r)} \\ \vdots & \vdots & \ddots & \vdots \\ f_{o(l_r)\_t}^{(1)} & f_{o(l_r)\_t}^{(2)} & \cdots & f_{o(l_r)\_t}^{(r)} \end{bmatrix}$$

$$\det(P_t) = \sum_{i=1}^{r} f_{o(l_1)\_t}^{(i)} * M_{1r} \neq 0, \forall t \in T$$

Figura 5.6 Resumen de restricciones para la solución de una sesión multicast

5.7 Ejemplos y Resultados

5.7.1 Ejemplo 1: Sesión multicast de 9 nodos

5.7.1.1 Grafos y Matrices del Sistema

La Figura 5.7 y la Tabla 5.2 muestran el grafo y la matriz de capacidades, respectivamente, de una sesión multicast de 9 nodos, con un nodo fuente ($s = 1$), tres sumideros ($T = \{7,8,9\}$) y flujo máximo $r = 2$ paquetes en $\mathbb{F}_2^l$. Por tener un flujo máximo de 2, los códigos que etiquetarán los paquetes estarán en el espacio $\mathbb{F}_2^2 \backslash \{0\}$. Además se observa un nodo codificador (nodo 5), que para el planteamiento del modelo no es trascendente. La Tabla 5.3 muestra los enlaces y sus tipos. De esta tabla también se deduce que los conjuntos de enlaces de salida y enlaces de llegada están constituidos por:

$$S = \{(1,2), (1,3), (1,4)\}$$
$$L = \{(2,7), (6,7), (3,9), (6,8), (4,8), (6,9)\}$$

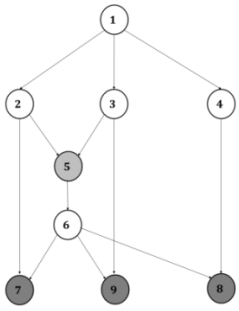


Figura 5.7 Sesión Multicast de 9 nodos

Tabla 5.2 Matriz de Adyacencia (Capacidad) C del Grafo G' de 9 nodos

nodos	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**
1	0	**1**	**1**	**1**	0	0	0	0	0
2	0	0	0	0	**1**	0	**1**	0	0
3	0	0	0	0	**1**	0	0	0	**1**
4	0	0	0	0	0	0	0	**1**	0
5	0	0	0	0	0	**1**	0	0	0
6	0	0	0	0	0	0	**1**	**1**	**1**
7	0	0	0	0	0	0	0	0	0
8	0	0	0	0	0	0	0	0	0
9	0	0	0	0	0	0	0	0	0

Tabla 5.3 Tabla de Enlaces

Índice	1	2	3	4	5	6	7	8	9	10	11	12
Origen	1	1	1	2	2	3	3	4	5	6	6	6
Destino	2	3	4	5	7	5	9	8	6	7	8	9
Tipo	1	1	1	0	2	0	2	2	0	2	2	2

El grafo lineal dirigido $\mathfrak{G}$ se presenta en la Figura 5.8:

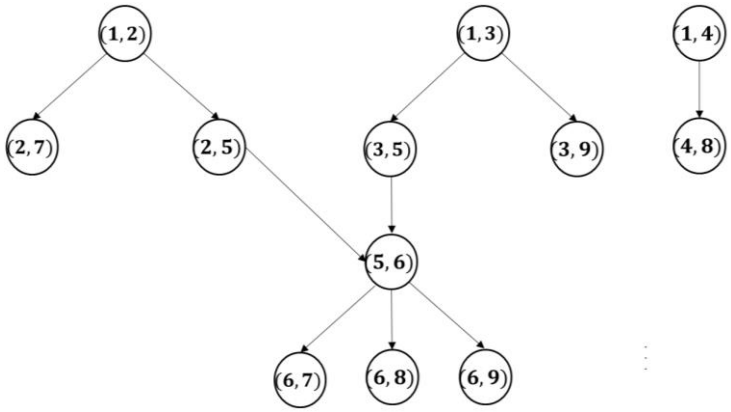


Figura 5.8 Grafo Lineal Dirigido para 9 nodos

Se observa que el camino más largo es de tres saltos, por lo cual solo se puede iterar hasta $n = 3$ y en $n = 4, F^4 = 0$. En la Tabla 5.4 hasta la Tabla 5.6, se muestran las iteraciones desde un salto hasta alcanzar los tres saltos en F^3. La Tabla 5.7 muestra la matriz de enlaces y la Tabla 5.8 muestra la matriz de enlaces reducida.

Tabla 5.4 Matriz de un salto F

Enlace	**(1,2)**	**(1,3)**	**(1,4)**	**(2,5)**	**(2,7)**	**(3,5)**	**(3,9)**	**(4,8)**	**(5,6)**	**(6,7)**	**(6,8)**	**(6,9)**
(1,2)	0	0	0	**1**	**1**	0	0	0	0	0	0	0
(1,3)	0	0	0	0	0	**1**	**1**	0	0	0	0	0
(1,4)	0	0	0	0	0	0	0	**1**	0	0	0	0
(2,5)	0	0	0	0	0	0	0	0	**1**	0	0	0
(2,7)	0	0	0	0	0	0	0	0	0	0	0	0
(3,5)	0	0	0	0	0	0	0	0	**1**	0	0	0
(3,8)	0	0	0	0	0	0	0	0	0	0	0	0
(4,9)	0	0	0	0	0	0	0	0	0	0	0	0
(5,6)	0	0	0	0	0	0	0	0	0	**1**	**1**	**1**
(6,7)	0	0	0	0	0	0	0	0	0	0	0	0
(6,8)	0	0	0	0	0	0	0	0	0	0	0	0
(6,9)	0	0	0	0	0	0	0	0	0	0	0	0

Tabla 5.5 Matriz de dos saltos F^2

Enlace	(1,2)	(1,3)	(1,4)	(2,5)	(2,7)	(3,5)	(3,9)	(4,8)	(5,6)	(6,7)	(6,8)	(6,9)
(1,2)	0	0	0	0	0	0	0	0	**1**	0	0	0
(1,3)	0	0	0	0	0	0	0	0	**1**	0	0	0
(1,4)	0	0	0	0	0	0	0	0	0	0	0	0
(2,5)	0	0	0	0	0	0	0	0	0	**1**	**1**	**1**
(2,7)	0	0	0	0	0	0	0	0	0	0	0	0
(3,5)	0	0	0	0	0	0	0	0	0	**1**	**1**	**1**
(3,8)	0	0	0	0	0	0	0	0	0	0	0	0
(4,9)	0	0	0	0	0	0	0	0	0	0	0	0
(5,6)	0	0	0	0	0	0	0	0	0	0	0	0
(6,7)	0	0	0	0	0	0	0	0	0	0	0	0
(6,8)	0	0	0	0	0	0	0	0	0	0	0	0
(6,9)	0	0	0	0	0	0	0	0	0	0	0	0

Tabla 5.6 Matriz de tres saltos F^3

Enlace	(1,2)	(1,3)	(1,4)	(2,5)	(2,7)	(3,5)	(3,9)	(4,8)	(5,6)	(6,7)	(6,8)	(6,9)
(1,2)	0	0	0	0	0	0	0	0	0	**1**	**1**	**1**
(1,3)	0	0	0	0	0	0	0	0	0	**1**	**1**	**1**
(1,4)	0	0	0	0	0	0	0	0	0	0	0	0
(2,5)	0	0	0	0	0	0	0	0	0	0	0	0
(2,7)	0	0	0	0	0	0	0	0	0	0	0	0
(3,5)	0	0	0	0	0	0	0	0	0	0	0	0
(3,8)	0	0	0	0	0	0	0	0	0	0	0	0
(4,9)	0	0	0	0	0	0	0	0	0	0	0	0
(5,6)	0	0	0	0	0	0	0	0	0	0	0	0
(6,7)	0	0	0	0	0	0	0	0	0	0	0	0
(6,8)	0	0	0	0	0	0	0	0	0	0	0	0
(6,9)	0	0	0	0	0	0	0	0	0	0	0	0

Tabla 5.7 Matriz de Enlaces Q

Enlace	(1,2)	(1,3)	(1,4)	(2,5)	(2,7)	(3,5)	(3,9)	(4,8)	(5,6)	(6,7)	(6,8)	(6,9)
(1,2)	0	0	0	0	**1**	0	0	0	0	**1**	**1**	**1**
(1,3)	0	0	0	0	0	0	**1**	0	0	**1**	**1**	**1**
(1,4)	0	0	0	0	0	0	0	**1**	0	0	0	0

Tabla 5.8 Matriz de Enlaces reducida Q_r

Enlace	(2,7)	(3,9)	(4,8)	(6,7)	(6,8)	(6,9)
(1,2)	**1**	0	0	**1**	**1**	**1**
(1,3)	0	**1**	0	**1**	**1**	**1**
(1,4)	0	0	**1**	0	0	0

5.7.1.2 Restricciones del sistema

De acuerdo con la sesión multicast representada en el grafo de la Figura 5.7, se muestra el siguiente conjunto de restricciones derivadas de la aplicación del modelo.

1. Restricciones en los flujos de enlaces de salida:

   Caso I: Flujos con paquetes simples:

$$f_{1\_2} = 1 \vee 2$$
$$f_{1\_3} = 1 \vee 2$$
$$f_{1\_4} = 1 \vee 2$$

Caso II: Flujos con paquetes combinados:

$$f_{1\_2} \neq 0$$
$$f_{1\_3} \neq 0$$
$$f_{1\_4} \neq 0$$

2. Restricciones en los flujos de enlaces de llegada:

   Con el apoyo de la matriz de enlaces reducida Q_r, se tiene:

$$f_{2\_7} = f_{1\_2} \qquad f_{6\_7} = f_{1\_2} + f_{1\_3}$$
$$f_{3\_9} = f_{1\_3} \qquad f_{6\_8} = f_{1\_2} + f_{1\_3}$$
$$f_{4\_8} = f_{1\_4} \qquad f_{6\_9} = f_{1\_2} + f_{1\_3}$$

3. Restricciones en los enlaces de llegada y en los nodos sumideros:

   a. Flujos de llegada no nulos:

$$f_{2\_7} \neq 0 \qquad f_{6\_7} \neq 0$$
$$f_{3\_9} \neq 0 \qquad f_{6\_8} \neq 0$$
$$f_{4\_8} \neq 0 \qquad f_{6\_9} \neq 0$$

   b. Combinación lineal de los flujos de llegada a un nodo sumidero no nula:

$$f_{2\_7} + f_{6\_7} \neq 0$$
$$f_{4\_8} + f_{6\_8} \neq 0$$
$$f_{3\_9} + f_{6\_9} \neq 0$$

   c. Matriz de códigos de etiquetas con tamaño $r = 2$:
   Los códigos de este espacio son:

$$\wp = \mathbb{F}_2^2 \backslash \{0\} = \{01,10,11\}$$

   A continuación, en la Tabla 5.9, se presentan las matrices de códigos y la restricción de linealidad independiente entre los flujos entrantes a cada nodo sumidero en T:

Tabla 5.9 Matrices de Código y Restricciones Linealmente Independientes

Para $t = 7$	**Para $t = 8$**	**Para $t = 9$**
$P_7 = \begin{bmatrix} f_{2\_7}^1 & f_{2\_7}^2 \\ f_{6\_7}^1 & f_{6\_7}^2 \end{bmatrix}$	$P_8 = \begin{bmatrix} f_{4_8}^1 & f_{4_8}^2 \\ f_{6_8}^1 & f_{6_8}^2 \end{bmatrix}$	$P_9 = \begin{bmatrix} f_{3\_9}^1 & f_{3\_9}^2 \\ f_{6\_9}^1 & f_{6\_9}^2 \end{bmatrix}$
$f_{2\_7}^1 * f_{6\_7}^2 + f_{2\_7}^2 * f_{6\_7}^1 \neq 0$	$f_{4\_8}^1 * f_{6\_8}^2 + f_{4\_8}^2 * f_{6\_8}^1 \neq 0$	$f_{3\_9}^1 * f_{6\_9}^2 + f_{3\_9}^2 * f_{6\_9}^1 \neq 0$

5.7.1.3 Solución al sistema

Al llevar este sistema a un solucionador de ecuaciones (en este trabajo se utilizó Z3[116], que es probador de teoremas y solucionador de lógica de primer orden, dentro de un código escrito en Python[117]), se logran las siguientes respuestas para cada valor de flujo (paquete) en los enlaces de salida y cada valor de flujo (paquete) en los enlaces de llegada:

Caso I: Flujos con paquetes simples:
Códigos de paquetes emergentes por enlaces de salida:

$$f_{1\_2} = 2 = B$$
$$f_{1\_3} = 1 = A$$
$$f_{1\_4} = 2 = B$$

Códigos de paquetes entrantes por los enlaces de llegada en la Tabla 5.10:

Tabla 5.10 Códigos de Paquetes Entrantes por los Enlaces de Llegada - Caso I

Para $t = 7$	**Para $t = 8$**	**Para $t = 9$**
$f_{2\_7} = 2 = B$	$f_{4\_8} = 2 = B$	$f_{3\_9} = 1 = A$
$f_{6\_7} = 3 = A + B$	$f_{6\_8} = 3 = A + B$	$f_{6\_9} = 3 = A + B$

El grafo de sesión multicast con la solución, se muestra en la Figura 5.9(a).

Caso II: Flujos con paquetes combinados:
Códigos de paquetes emergentes por enlaces de salida:

$$f_{1\_2} = 1 = A$$
$$f_{1\_3} = 3 = A + B$$
$$f_{1\_4} = 3 = A + B$$

Códigos de paquetes entrantes por los enlaces de llegada en la Tabla 5.11:

Tabla 5.11 Códigos de Paquetes Entrantes por los Enlaces de Llegada - Caso II

Para $t = 7$	**Para $t = 8$**	**Para $t = 9$**
$f_{2\_7} = 1 = A$	$f_{4\_8} = 3 = A + B$	$f_{3\_9} = 3 = A + B$
$f_{6\_7} = 2 = B$	$f_{6\_8} = 2 = B$	$f_{6\_9} = 2 = B$

El grafo de sesión multicast con la solución se muestra en la Figura 5.9(b)

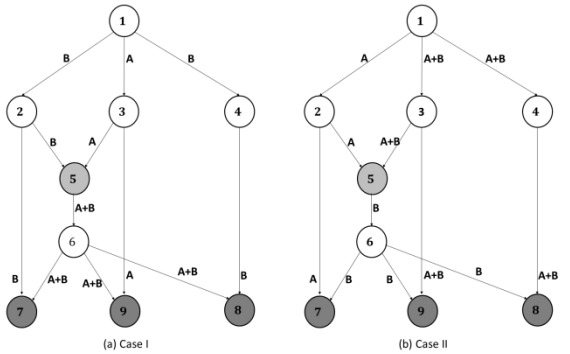


Figura 5.9 Solución en grafo G' de 9 nodos

5.7.2 Ejemplo 2: Sesión multicast de 18 nodos

5.7.2.1 Grafos y Matrices del Sistema

La Figura 5.10 y la Tabla 5.12 muestran el grafo y la matriz de capacidades, respectivamente, de

una sesión multicast de 18 nodos, con un nodo fuente ($s = 1$), tres sumideros ($T = \{18,19,20\}$ y flujo máximo $r = 4$ paquetes en $\mathbb{F}_2^l$. Por tener un flujo máximo de 4, los códigos que etiquetarán los paquetes estarán en el espacio $\mathbb{F}_2^4 \backslash \{0\}$. Además se observan dos nodos codificadores (nodos 8 y 12) que para el planteamiento del modelo no son relevantes.

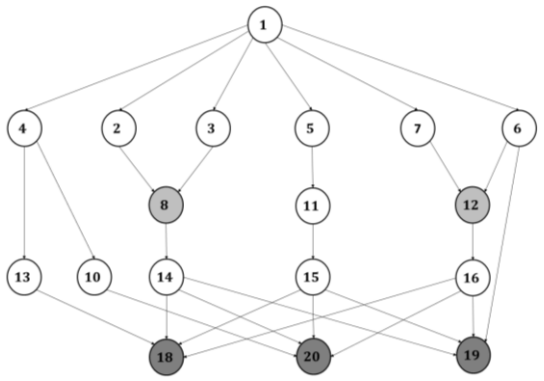


Figura 5.10 Sesión Multicast de 18 nodos

Tabla 5.12 Matriz de Adyacencia (Capacidad) C del Grafo G' de 18 nodos

nodos	1	2	3	4	5	6	7	8	10	11	12	13	14	15	16	18	19	20
1	0	**1**	**1**	**1**	**1**	**1**	**1**	0	0	0	0	0	0	0	0	0	0	0
2	0	0	0	0	0	0	0	**1**	0	0	0	0	0	0	0	0	0	0
3	0	0	0	0	0	0	0	**1**	0	0	0	0	0	0	0	0	0	0
4	0	0	0	0	0	0	0	0	**1**	0	0	**1**	0	0	0	0	0	0
5	0	0	0	0	0	0	0	0	0	**1**	0	0	0	0	0	0	0	0
6	0	0	0	0	0	0	0	0	0	0	**1**	0	0	0	0	0	**1**	0
7	0	0	0	0	0	0	0	0	0	0	**1**	0	0	0	0	0	0	0
8	0	0	0	0	0	0	0	0	0	0	0	0	**1**	0	0	0	0	0
10	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	**1**
11	0	0	0	0	0	0	0	0	0	0	0	0	0	**1**	0	0	0	0
12	0	0	0	0	0	0	0	0	0	0	0	0	0	0	**1**	0	0	0
13	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	**1**	0	0
14	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	**1**	**1**	**1**
15	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	**1**	**1**	**1**
16	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	**1**	**1**	**1**
18	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
19	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
20	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

La Tabla 5.13 muestra los enlaces y sus tipos. De esta tabla también se deduce que los conjuntos de enlaces de salida y enlaces de llegada están constituidos por:

$$S = \{(1,2), (1,3), (1,4), (1,5), (1,6), (1,7)\}$$

$$L = \left\{\begin{matrix}(13,18), (14,18), (15,18), (16,18), (6,19), (14,19), (15,19), (16,19), (10,20),\\ (14,20), (15,20), (16,20)\end{matrix}\right\}$$

Tabla 5.13 Tabla de Enlaces

Índice	1	2	3	4	5	6	7	8	9	10	11	12	13	14
Origen	1	1	1	1	1	1	2	3	4	4	5	6	6	7
Destino	2	3	4	5	6	7	8	8	10	13	11	12	19	12
Tipo	1	1	1	1	1	1	0	0	0	0	0	0	2	0

Índice	15	16	17	18	19	20	21	22	23	24	25	26	27	28

Origen	8	10	11	12	13	14	14	14	15	15	15	16	16	16
Destino	14	20	15	16	18	18	19	20	18	19	20	18	19	20
Tipo	0	2	0	0	2	2	2	2	2	2	2	2	2	2

El grafo lineal dirigido $\mathfrak{G}$ se presenta en la Figura 5.11:

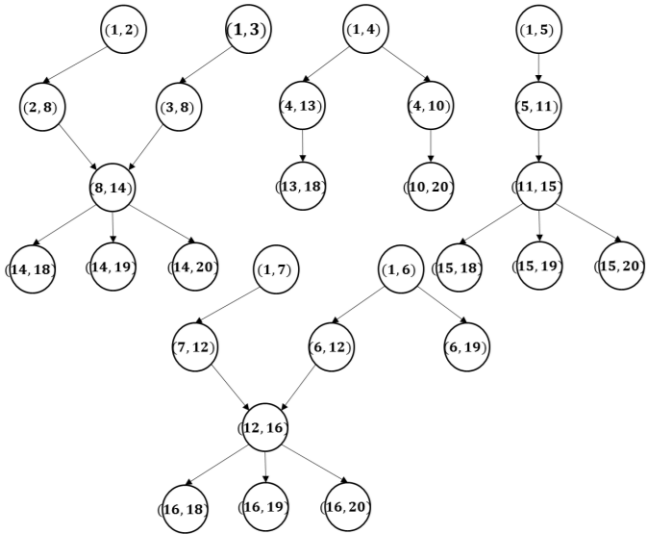


Figura 5.11 Grafo lineal dirigido para 18 nodos

Se observa que el camino más largo es de tres saltos, por lo cual solo se puede iterar hasta $n = 3$ y en $n = 4, F^4 = 0$. En la Tabla 5.14 hasta la Tabla 5.16, se muestran las iteraciones desde un salto hasta alcanzar los tres saltos en F^3. La Tabla 5.17 muestra la matriz de enlaces y la Tabla 5.18 muestra la matriz de enlaces reducida.

5.7.2.2 Restricciones del sistema

De acuerdo con la sesión multicast representada en el grafo de la Figura 5.10, se muestra el siguiente conjunto de restricciones derivadas de la aplicación del modelo.

1. Restricciones en los flujos de enlaces de salida:

   Caso I: Flujos con paquetes simples:

$$f_{1\_2} = 1\text{V}2\text{V}4\text{V}8 \qquad f_{1\_5} = 1\text{V}2\text{V}4\text{V}8$$
$$f_{1\_3} = 1\text{V}2\text{V}4\text{V}8 \qquad f_{1\_6} = 1\text{V}2\text{V}4\text{V}8$$
$$f_{1\_4} = 1\text{V}2\text{V}4\text{V}8 \qquad f_{1\_7} = 1\text{V}2\text{V}4\text{V}8$$

   Caso II: Flujos con paquetes combinados:

$$f_{1\_2} \neq 0 \qquad f_{1\_5} \neq 0$$
$$f_{1\_3} \neq 0 \qquad f_{1\_6} \neq 0$$
$$f_{1\_4} \neq 0 \qquad f_{1\_7} \neq 0$$

Tabla 5.14 Matriz de un salto *F*

Enlaces	**(1,2)**	**(1,3)**	**(1,4)**	**(1,5)**	**(1,6)**	**(1,7)**	**(2,8)**	**(3,8)**	**(4,10)**	**(4,13)**	**(5,11)**	**(6,12)**	**(6,19)**	**(7,12)**	**(8,14)**	**(10,20)**	**(11,15)**	**(12,16)**	**(13,18)**	**(14,18)**	**(14,19)**	**(14,20)**	**(15,18)**	**(15,19)**	**(15,20)**	**(16,18)**	**(16,19)**	**(16,20)**
(1,2)	0	0	0	0	0	0	**1**	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
(1,3)	0	0	0	0	0	0	0	**1**	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
(1,4)	0	0	0	0	0	0	0	0	**1**	**1**	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
(1,5)	0	0	0	0	0	0	0	0	0	0	**1**	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
(1,6)	0	0	0	0	0	0	0	0	0	0	0	**1**	**1**	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
(1,7)	0	0	0	0	0	0	0	0	0	0	0	0	0	**1**	0	0	0	0	0	0	0	0	0	0	0	0	0	0
(2,8)	0	0	0	0	0	0	0	0	0	0	0	0	0	0	**1**	0	0	0	0	0	0	0	0	0	0	0	0	0
(3,8)	0	0	0	0	0	0	0	0	0	0	0	0	0	0	**1**	0	0	0	0	0	0	0	0	0	0	0	0	0
(4,10)	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	**1**	0	0	0	0	0	0	0	0	0	0	0	0
(4,13)	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	**1**	0	0	0	0	0	0	0	0	0	0
(5,11)	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	**1**	0	0	0	0	0	0	0	0	0	0	0
(6,12)	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	**1**	0	0	0	0	0	0	0	0	0	0
(6,19)	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
(7,12)	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	**1**	0	0	0	0	0	0	0	0	0	0
(8,14)	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	**1**	**1**	**1**	0	0	0	0	0	0	0
(10,20)	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
(11,15)	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	**1**	**1**	**1**	0	0	0	0
(12,16)	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	**1**	**1**	**1**	**1**
(13,18)	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
(14,18)	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
(14,19)	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
(14,20)	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
(15,18)	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
(15,19)	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
(15,20)	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
(16,18)	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
(16,19)	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
(16,20)	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Tabla 5.15 Matriz de dos saltos F^2

Enlaces	**(1,2)**	**(1,3)**	**(1,4)**	**(1,5)**	**(1,6)**	**(1,7)**	**(2,8)**	**(3,8)**	**(4,10)**	**(4,13)**	**(5,11)**	**(6,12)**	**(6,19)**	**(7,12)**	**(8,14)**	**(10,20)**	**(11,15)**	**(12,16)**	**(13,18)**	**(14,18)**	**(14,19)**	**(14,20)**	**(15,18)**	**(15,19)**	**(15,20)**	**(16,18)**	**(16,19)**	**(16,20)**
(1,2)	0	0	0	0	0	0	0	0	0	0	0	0	0	0	**1**	0	0	0	0	0	0	0	0	0	0	0	0	0
(1,3)	0	0	0	0	0	0	0	0	0	0	0	0	0	0	**1**	0	0	0	0	0	0	0	0	0	0	0	0	0
(1,4)	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	**1**	0	0	**1**	0	0	0	0	0	0	0	0	0
(1,5)	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	**1**	0	0	0	0	0	0	0	0	0	0	0
(1,6)	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	**1**	0	0	0	0	0	0	0	0	0	0
(1,7)	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	**1**	0	0	0	0	0	0	0	0	0	0
(2,8)	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	**1**	**1**	**1**	0	0	0	0	0	0
(3,8)	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	**1**	**1**	**1**	0	0	0	0	0	0
(4,10)	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
(4,13)	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
(5,11)	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	**1**	**1**	**1**	0	0	0
(6,12)	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	**1**	**1**	**1**
(6,19)	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
(7,12)	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	**1**	**1**	**1**
(8,14)	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
(10,20)	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
(11,15)	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
(12,16)	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
(13,18)	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
(14,18)	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
(14,19)	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
(14,20)	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
(15,18)	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
(15,19)	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
(15,20)	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
(16,18)	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
(16,19)	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
(16,20)	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Tabla 5.16 Matriz de tres saltos F^3

Enlaces	**(1,2)**	**(1,3)**	**(1,4)**	**(1,5)**	**(1,6)**	**(1,7)**	**(2,8)**	**(3,8)**	**(4,10)**	**(4,13)**	**(5,11)**	**(6,12)**	**(6,19)**	**(7,12)**	**(8,14)**	**(10,20)**	**(11,15)**	**(12,16)**	**(13,18)**	**(14,18)**	**(14,19)**	**(14,20)**	**(15,18)**	**(15,19)**	**(15,20)**	**(16,18)**	**(16,19)**	**(16,20)**
(1,2)	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	**1**	**1**	**1**	0	0	0	0	0	0
(1,3)	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	**1**	**1**	**1**	0	0	0	0	0	0
(1,4)	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
(1,5)	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	**1**	**1**	**1**	0	0	0
(1,6)	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	**1**	**1**	**1**
(1,7)	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	**1**	**1**	**1**
(2,8)	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
(3,8)	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
(4,10)	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
(4,13)	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
(5,11)	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
(6,12)	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
(6,19)	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
(7,12)	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
(8,14)	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
(10,20)	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
(11,15)	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
(12,16)	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
(13,18)	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
(14,18)	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
(14,19)	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
(14,20)	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
(15,18)	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
(15,19)	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
(15,20)	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
(16,18)	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
(16,19)	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
(16,20)	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Tabla 5.17 Matriz de Enlaces Q

Enlaces	**(1,2)**	**(1,3)**	**(1,4)**	**(1,5)**	**(1,6)**	**(1,7)**	**(2,8)**	**(3,8)**	**(4,10)**	**(4,13)**	**(5,11)**	**(6,12)**	**(6,19)**	**(7,12)**	**(8,14)**	**(10,20)**	**(11,15)**	**(12,16)**	**(13,18)**	**(14,18)**	**(14,19)**	**(14,20)**	**(15,18)**	**(15,19)**	**(15,20)**	**(16,18)**	**(16,19)**	**(16,20)**
(1,2)	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	**1**	**1**	**1**	0	0	0	0	0	0
(1,3)	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	**1**	**1**	**1**	0	0	0	0	0	0
(1,4)	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	**1**	0	0	**1**	0	0	0	0	0	0	0	0	0
(1,5)	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	**1**	**1**	**1**	0	0	0
(1,6)	0	0	0	0	0	0	0	0	0	0	0	0	**1**	0	0	0	0	0	0	0	0	0	0	0	0	**1**	**1**	**1**
(1,7)	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	**1**	**1**	**1**

Tabla 5.18 Matriz de Enlaces reducida Q_r

Enlaces	**(6,19)**	**(10,20)**	**(13,18)**	**(14,18)**	**(14,19)**	**(14,20)**	**(15,18)**	**(15,19)**	**(15,20)**	**(16,18)**	**(16,19)**	**(16,20)**
(1,2)	0	0	0	**1**	**1**	**1**	0	0	0	0	0	0
(1,3)	0	0	0	**1**	**1**	**1**	0	0	0	0	0	0
(1,4)	0	**1**	**1**	0	0	0	0	0	0	0	0	0
(1,5)	0	0	0	0	0	0	**1**	**1**	**1**	0	0	0
(1,6)	**1**	0	0	0	0	0	0	0	0	**1**	**1**	**1**
(1,7)	0	0	0	0	0	0	0	0	0	**1**	**1**	**1**

2. Restricciones en los flujos de enlaces de llegada:

   Con el apoyo de la matriz de enlaces reducida Q_r, se tiene:

$$f_{13\_18} = f_{1\_4} \qquad f_{6\_19} = f_{1\_6} \qquad f_{10\_20} = f_{1\_4}$$
$$f_{14\_18} = f_{1\_2} + f_{1\_3} \qquad f_{14\_19} = f_{1\_2} + f_{1\_3} \qquad f_{14\_20} = f_{1\_2} + f_{1\_3}$$
$$f_{15\_18} = f_{1\_5} \qquad f_{15\_19} = f_{1\_5} \qquad f_{15\_20} = f_{1\_5}$$
$$f_{16\_18} = f_{1\_6} + f_{1\_7} \qquad f_{16\_19} = f_{1\_6} + f_{1\_7} \qquad f_{16\_20} = f_{1\_6} + f_{1\_7}$$

3. Restricciones en los enlaces de llegada y en los nodos sumideros:

   a. Flujos de llegada no nulos:

$$f_{13\_18} \neq 0 \qquad f_{6\_19} \neq 0 \qquad f_{10\_20} \neq 0$$
$$f_{14\_18} \neq 0 \qquad f_{14\_19} \neq 0 \qquad f_{14\_20} \neq 0$$
$$f_{15\_18} \neq 0 \qquad f_{15\_19} \neq 0 \qquad f_{15\_20} \neq 0$$
$$f_{16\_18} \neq 0 \qquad f_{16\_19} \neq 0 \qquad f_{16\_20} \neq 0$$

   b. Combinación lineal de los flujos de llegada a un nodo sumidero no nula:

$$f_{13\_18} + f_{14\_18} + f_{15\_18} + f_{16\_18} \neq 0$$
$$f_{6\_19} + f_{14\_19} + f_{15\_19} + f_{16\_19} \neq 0$$
$$f_{10\_20} + f_{14\_20} + f_{15\_20} + f_{16\_20} \neq 0$$

   c. Matriz de códigos de etiquetas con tamaño $r = 4$:
   Los códigos de este espacio son:
   $\wp = \mathbb{F}_2^4 \backslash \{0\} = \{0001,0010,0011,0100,0101,0110,0111,1000,1001,1010,1011,1100,1101,1110,1111\}$

   A continuación se presentan las matrices de códigos y la restricción de linealidad independiente entre los flujos entrantes a cada nodo sumidero en T:

   Para $t = 18$:

$$P_{18} = \begin{bmatrix} f^1_{13\_18} & f^2_{13\_18} & f^3_{13\_18} & f^4_{13\_18} \\ f^1_{14\_18} & f^2_{14\_18} & f^3_{14\_18} & f^4_{14\_18} \\ f^1_{15\_18} & f^2_{15\_18} & f^3_{15\_18} & f^4_{15\_18} \\ f^1_{16\_18} & f^2_{16\_18} & f^3_{16\_18} & f^4_{16\_18} \end{bmatrix}$$

$$f^1_{13\_18} * (f^2_{14\_18} * (f^3_{15\_18} * f^4_{16\_18} + f^4_{15\_18} * f^3_{16\_18}) + f^3_{14\_18} * (f^2_{15\_18} * f^4_{16\_18} + f^4_{15\_18} * f^2_{16\_18}) + f^4_{14\_18} * (f^2_{15\_18} * f^3_{16\_18} + f^3_{15\_18} * f^2_{16\_18})) + f^2_{13\_18} * (f^1_{14\_18} * (f^3_{15\_18} * f^4_{16\_18} + f^4_{15\_18} * f^3_{16\_18}) + f^3_{14\_18} * (f^1_{15\_18} * f^4_{16\_18} + f^4_{15\_18} * f^1_{16\_18}) + f^4_{14\_18} * (f^1_{15\_18} * f^3_{16\_18} + f^3_{15\_18} * f^1_{16\_18})) + f^3_{13\_18} * (f^1_{14\_18} * (f^2_{15\_18} * f^4_{16\_18} + f^4_{15\_18} * f^2_{16\_18}) + f^2_{14\_18} * (f^1_{15\_18} * f^4_{16\_18} + f^4_{15\_18} * f^1_{16\_18}) + f^4_{14\_18} * (f^1_{15\_18} * f^2_{16\_18} + f^2_{15\_18} * f^1_{16\_18})) + f^4_{13\_18} * (f^1_{14\_18} * (f^2_{15\_18} * f^3_{16\_18} + f^3_{15\_18} * f^2_{16\_18}) + f^2_{14\_18} * (f^1_{15\_18} * f^3_{16\_18} + f^3_{15\_18} * f^1_{16\_18}) + f^3_{14\_18} * (f^1_{15\_18} * f^2_{16\_18} + f^2_{15\_18} * f^1_{16\_18})) \neq 0$$

   Para $t = 19$:

$$P_{19} = \begin{bmatrix} f^1_{6\_19} & f^2_{6\_19} & f^3_{6\_19} & f^4_{6\_19} \\ f^1_{14\_19} & f^2_{14\_19} & f^3_{14\_19} & f^4_{14\_19} \\ f^1_{15\_19} & f^2_{15\_19} & f^3_{15\_19} & f^4_{15\_19} \\ f^1_{16\_19} & f^2_{16\_19} & f^3_{16\_19} & f^4_{16\_19} \end{bmatrix}$$

$f^1_{6\_19} * (f^2_{14\_19} * (f^3_{15\_19} * f^4_{16\_19} + f^4_{15\_19} * f^3_{16\_19}) + f^3_{14\_19} * (f^2_{15\_19} * f^4_{16\_19} + f^4_{15\_19} * f^2_{16\_19}) + f^4_{14\_19} * (f^2_{15\_19} * f^3_{16\_19} + f^3_{15\_19} * f^2_{16\_19})) + f^2_{6\_19} * (f^1_{14\_19} * (f^3_{15\_19} * f^4_{16\_19} + f^4_{15\_19} * f^3_{16\_19}) + f^3_{14\_19} * (f^1_{15\_19} * f^4_{16\_19} + f^4_{15\_19} * f^1_{16\_19}) + f^4_{14\_19} * (f^1_{15\_19} * f^3_{16\_19} + f^3_{15\_19} * f^1_{16\_19})) + f^3_{6\_19} * (f^1_{14\_19} * (f^2_{15\_19} * f^4_{16\_19} + f^4_{15\_19} * f^2_{16\_19}) + f^2_{14\_19} * (f^1_{15\_19} * f^4_{16\_19} + f^4_{15\_19} * f^1_{16\_19}) + f^4_{14\_19} * (f^1_{15\_19} * f^2_{16\_19} + f^2_{15\_19} * f^1_{16\_19})) + f^4_{6\_19} * (f^1_{14\_19} * (f^2_{15\_19} * f^3_{16\_19} + f^3_{15\_19} * f^2_{16\_19}) + f^2_{14\_19} * (f^1_{15\_19} * f^3_{16\_19} + f^3_{15\_19} * f^1_{16\_19}) + f^3_{14\_19} * (f^1_{15\_19} * f^2_{16\_19} + f^2_{15\_19} * f^1_{16\_19})) \neq 0$

Para $t = 20$:

$$P_{20} = \begin{bmatrix} f^1_{10\_20} & f^2_{10\_20} & f^3_{10\_20} & f^4_{10\_20} \\ f^1_{14\_20} & f^2_{14\_20} & f^3_{14\_20} & f^4_{14\_20} \\ f^1_{15\_20} & f^2_{15\_20} & f^3_{15\_20} & f^4_{15\_20} \\ f^1_{16\_20} & f^2_{16\_20} & f^3_{16\_20} & f^4_{16\_20} \end{bmatrix}$$

$f^1_{10\_20} * (f^2_{14\_20} * (f^3_{15\_20} * f^4_{16\_20} + f^4_{15\_20} * f^3_{16\_20}) + f^3_{14\_20} * (f^2_{15\_20} * f^4_{16\_20} + f^4_{15\_20} * f^2_{16\_20}) + f^4_{14\_20} * (f^2_{15\_20} * f^3_{16\_20} + f^3_{15\_20} * f^2_{16\_20})) + f^2_{10\_20} * (f^1_{14\_20} * (f^3_{15\_20} * f^4_{16\_20} + f^4_{15\_20} * f^3_{16\_20}) + f^3_{14\_20} * (f^1_{15\_20} * f^4_{16\_20} + f^4_{15\_20} * f^1_{16\_20}) + f^4_{14\_20} * (f^1_{15\_20} * f^3_{16\_20} + f^3_{15\_20} * f^1_{16\_20})) + f^3_{10\_20} * (f^1_{14\_20} * (f^2_{15\_20} * f^4_{16\_20} + f^4_{15\_20} * f^2_{16\_20}) + f^2_{14\_20} * (f^1_{15\_20} * f^4_{16\_20} + f^4_{15\_20} * f^1_{16\_20}) + f^4_{14\_20} * (f^1_{15\_20} * f^2_{16\_20} + f^2_{15\_20} * f^1_{16\_20})) + f^4_{10\_20} * (f^1_{14\_20} * (f^2_{15\_20} * f^3_{16\_20} + f^3_{15\_20} * f^2_{16\_20}) + f^2_{14\_20} * (f^1_{15\_20} * f^3_{16\_20} + f^3_{15\_20} * f^1_{16\_20}) + f^3_{14\_20} * (f^1_{15\_20} * f^2_{16\_20} + f^2_{15\_20} * f^1_{16\_20})) \neq 0$

5.7.2.3 Solución al sistema

Se logran las siguientes respuestas para cada valor de flujo (paquete) en los enlaces de salida y cada valor de flujo (paquete) en los enlaces de llegada:

Caso I: Flujos con paquetes simples:
Códigos de paquetes emergentes por enlaces de salida:

$f_{1\_2} = 1 = A$
$f_{1\_3} = 8 = D$
$f_{1\_4} = 1 = A$
$f_{1\_5} = 4 = C$
$f_{1\_6} = 2 = B$
$f_{1\_7} = 1 = A$

Códigos de paquetes entrantes por los enlaces de llegada en la Tabla 5.19:

Tabla 5.19 Códigos de Paquetes Entrantes por los Enlaces de Llegada - Caso I

Para $t = 18$	**Para $t = 19$**	**Para $t = 20$**
$f_{13\_18} = 1 = A$	$f_{6\_19} = 2 = B$	$f_{10\_20} = 1 = A$
$f_{14\_18} = 9 = A + D$	$f_{14\_19} = 9 = A + D$	$f_{14\_20} = 9 = A + D$
$f_{15\_18} = 4 = C$	$f_{15\_19} = 4 = C$	$f_{15\_20} = 4 = C$
$f_{16\_18} = 3 = A + B$	$f_{16\_19} = 3 = A + B$	$f_{16\_20} = 3 = A + B$

El grafo de sesión multicast con la solución se muestra en la Figura 5.12(a).

Caso II: Flujos con paquetes combinados:
Códigos de paquetes emergentes por enlaces de salida:

$f_{1\_2} = 7 = A + B + C$
$f_{1\_3} = 1 = A$
$f_{1\_4} = 3 = A + B$
$f_{1\_5} = 12 = C + D$
$f_{1\_6} = 3 = A + B$
$f_{1\_7} = 11 = A + B + D$

Códigos de paquetes entrantes por los enlaces de llegada en la Tabla 5.20:

Tabla 5.20 Códigos de Paquetes Entrantes por los Enlaces de Llegada - Caso II

Para $t = 18$	**Para $t = 19$**	**Para $t = 20$**
$f_{13\_18} = 3 = A + B$	$f_{6\_19} = 3 = A + B$	$f_{10\_20} = 3 = A + B$
$f_{14\_18} = 6 = B + C$	$f_{14\_19} = 6 = B + C$	$f_{14\_20} = 6 = B + C$
$f_{15\_18} = 12 = C + D$	$f_{15\_19} = 12 = C + D$	$f_{15\_20} = 12 = C + D$
$f_{16\_18} = 8 = D$	$f_{16\_19} = 8 = D$	$f_{16\_20} = 8 = D$

El grafo de sesión multicast con la solución se muestra en la Figura 5.12(b).

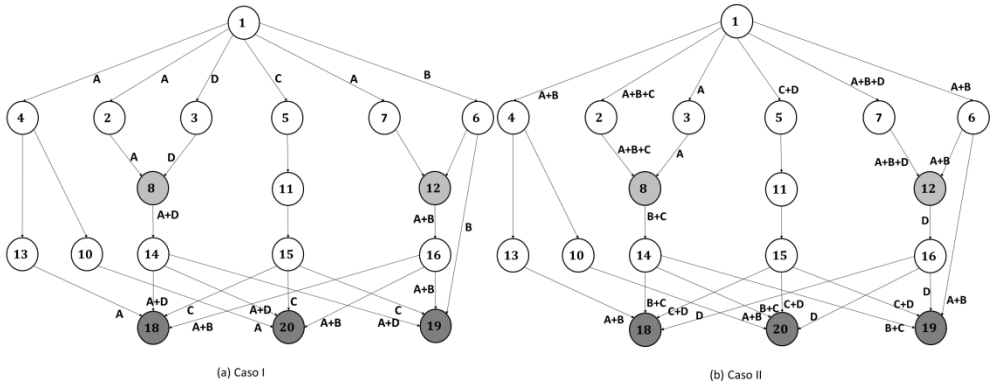


Figura 5.12 Solución en grafo G' de 18 nodos

5.7.3 Ejemplo 3: Sesión multicast de 11 nodos

5.7.3.1 Grafos y Matrices del Sistema

La Figura 5.13 y la Tabla 5.21 muestran el grafo y la matriz de capacidades, respectivamente, de una sesión multicast de 11 nodos, con un nodo fuente ($s = 1$), tres sumideros ($T = \{6,10,11\}$) y flujo máximo $r = 2$ paquetes en $\mathbb{F}_2^l$. Por tener un flujo máximo de 2, los códigos que etiquetarán los paquetes estarán en el espacio $\mathbb{F}_2^2 \backslash \{0\}$. Además se observan dos nodos codificadores (nodos 4,8), que no son transcendentes para el planteamiento del modelo.

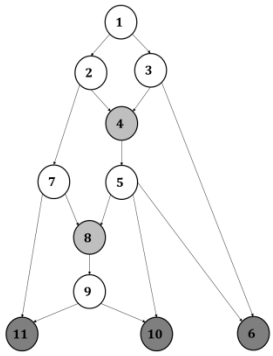


Figura 5.13 Sesión Multicast de 11 nodos

Tabla 5.21 Matriz de Adyacencia (Capacidad) C del Grafo G' de 11 nodos

nodos	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**
1	0	**1**	**1**	0	0	0	0	0	0	0	0
2	0	0	0	**1**	0	0	**1**	0	0	0	0
3	0	0	0	**1**	0	**1**	0	0	0	0	0
4	0	0	0	0	**1**	0	0	0	0	0	0
5	0	0	0	0	0	**1**	0	**1**	0	**1**	0
6	0	0	0	0	0	0	0	0	0	0	0
7	0	0	0	0	0	0	0	**1**	0	0	**1**
8	0	0	0	0	0	0	0	0	**1**	0	0
9	0	0	0	0	0	0	0	0	0	**1**	**1**
10	0	0	0	0	0	0	0	0	0	0	0
11	0	0	0	0	0	0	0	0	0	0	0

La Tabla 5.22 muestra los enlaces y sus tipos. De esta tabla también se deduce que los conjuntos de enlaces de salida y enlaces de llegada están constituidos por:

$$S = \{(1,2), (1,3)\}$$
$$L = \{(3,6), (5,6), (5,10), (9,10), (7,11), (9,11)\}$$

Tabla 5.22 Tabla de Enlaces

Índice	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Origen	1	1	2	2	3	3	4	5	5	5	7	7	8	9	9
Destino	2	3	4	7	4	6	5	6	8	10	8	11	9	10	11
Tipo	1	1	0	0	0	2	0	2	0	2	0	2	0	2	2

El grafo lineal dirigido $\mathfrak{G}$ se presenta a continuación:

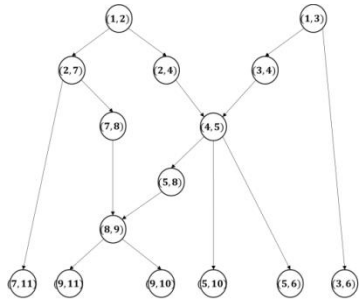


Figura 5.14 Grafo Lineal Dirigido para 11 nodos

Se observa que el camino más largo es de cinco saltos, por lo cual solo se puede iterar hasta $n = 5$ y en $n = 6, F^6 = 0$. En la Tabla 5.23 hasta la Tabla 5.27, se muestran las iteraciones desde un salto hasta alcanzar los tres saltos en F^5. La Tabla 5.28 muestra la matriz de enlaces y la Tabla 5.29 muestra la matriz de enlaces reducida.

Tabla 5.23 Matriz de un salto F

Enlace	(1,2)	(1,3)	(2,4)	(2,7)	(3,4)	(3,6)	(4,5)	(5,6)	(5,8)	(5,10)	(7,8)	(7,11)	(8,9)	(9,10)	(9,11)
(1,2)	0	0	**1**	**1**	0	0	0	0	0	0	0	0	0	0	0
(1,3)	0	0	0	0	**1**	**1**	0	0	0	0	0	0	0	0	0
(2,4)	0	0	0	0	0	0	**1**	0	0	0	0	0	0	0	0
(2,7)	0	0	0	0	0	0	0	0	0	0	**1**	**1**	0	0	0
(3,4)	0	0	0	0	0	0	**1**	0	0	0	0	0	0	0	0
(3,6)	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
(4,5)	0	0	0	0	0	0	0	**1**	**1**	**1**	0	0	0	0	0
(5,6)	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
(5,8)	0	0	0	0	0	0	0	0	0	0	0	0	**1**	0	0
(5,10)	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
(7,8)	0	0	0	0	0	0	0	0	0	0	0	0	**1**	0	0
(7,11)	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
(8,9)	0	0	0	0	0	0	0	0	0	0	0	0	0	**1**	**1**
(9,10)	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
(9,11)	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Tabla 5.24 Matriz de dos saltos F^2

Enlace	(1,2)	(1,3)	(2,4)	(2,7)	(3,4)	(3,6)	(4,5)	(5,6)	(5,8)	(5,10)	(7,8)	(7,11)	(8,9)	(9,10)	(9,11)
(1,2)	0	0	0	0	0	0	**1**	0	0	0	**1**	**1**	0	0	0
(1,3)	0	0	0	0	0	0	**1**	0	0	0	0	0	0	0	0
(2,4)	0	0	0	0	0	0	0	**1**	**1**	**1**	0	0	0	0	0
(2,7)	0	0	0	0	0	0	0	0	0	0	0	0	**1**	0	0
(3,4)	0	0	0	0	0	0	0	**1**	**1**	**1**	0	0	0	0	0
(3,6)	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
(4,5)	0	0	0	0	0	0	0	0	0	0	0	0	**1**	0	0
(5,6)	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
(5,8)	0	0	0	0	0	0	0	0	0	0	0	0	0	**1**	**1**
(5,10)	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
(7,8)	0	0	0	0	0	0	0	0	0	0	0	0	0	**1**	**1**
(7,11)	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
(8,9)	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
(9,10)	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
(9,11)	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Tabla 5.25 Matriz de tres saltos F^3

Enlace	(1,2)	(1,3)	(2,4)	(2,7)	(3,4)	(3,6)	(4,5)	(5,6)	(5,8)	(5,10)	(7,8)	(7,11)	(8,9)	(9,10)	(9,11)
(1,2)	0	0	0	0	0	0	0	**1**	**1**	**1**	0	0	**1**	0	0
(1,3)	0	0	0	0	0	0	0	**1**	**1**	**1**	0	0	0	0	0
(2,4)	0	0	0	0	0	0	0	0	0	0	0	0	**1**	0	0
(2,7)	0	0	0	0	0	0	0	0	0	0	0	0	0	**1**	**1**
(3,4)	0	0	0	0	0	0	0	0	0	0	0	0	**1**	0	0
(3,6)	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
(4,5)	0	0	0	0	0	0	0	0	0	0	0	0	0	**1**	**1**
(5,6)	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
(5,8)	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
(5,10)	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
(7,8)	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
(7,11)	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
(8,9)	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
(9,10)	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
(9,11)	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Tabla 5.26 Matriz de cuatro saltos F^4

Enlace	(1,2)	(1,3)	(2,4)	(2,7)	(3,4)	(3,6)	(4,5)	(5,6)	(5,8)	(5,10)	(7,8)	(7,11)	(8,9)	(9,10)	(9,11)
(1,2)	0	0	0	0	0	0	0	0	0	0	0	0	**1**	**1**	**1**
(1,3)	0	0	0	0	0	0	0	0	0	0	0	0	**1**	0	0
(2,4)	0	0	0	0	0	0	0	0	0	0	0	0	0	**1**	**1**
(2,7)	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
(3,4)	0	0	0	0	0	0	0	0	0	0	0	0	0	**1**	**1**
(3,6)	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
(4,5)	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
(5,6)	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
(5,8)	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
(5,10)	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
(7,8)	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
(7,11)	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
(8,9)	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
(9,10)	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
(9,11)	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Tabla 5.27 Matriz de cinco saltos F^5

Enlace	(1,2)	(1,3)	(2,4)	(2,7)	(3,4)	(3,6)	(4,5)	(5,6)	(5,8)	(5,10)	(7,8)	(7,11)	(8,9)	(9,10)	(9,11)
(1,2)	0	0	0	0	0	0	0	0	0	0	0	0	0	**1**	**1**
(1,3)	0	0	0	0	0	0	0	0	0	0	0	0	0	**1**	**1**
(2,4)	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
(2,7)	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
(3,4)	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
(3,6)	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
(4,5)	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
(5,6)	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
(5,8)	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
(5,10)	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
(7,8)	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
(7,11)	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
(8,9)	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
(9,10)	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
(9,11)	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Tabla 5.28 Matriz de Enlaces Q

Enlace	(1,2)	(1,3)	(2,4)	(2,7)	(3,4)	(3,6)	(4,5)	(5,6)	(5,8)	(5,10)	(7,8)	(7,11)	(8,9)	(9,10)	(9,11)
(1,2)	0	0	0	0	0	0	0	**1**	0	**1**	0	**1**	0	**2**	**2**
(1,3)	0	0	0	0	0	**1**	0	**1**	0	**1**	0	0	0	**1**	**1**

Tabla 5.29 Matriz de Enlaces reducida Q_r

Enlace	(3,6)	(5,6)	(5,10)	(7,11)	(9,10)	(9,11)
(1,2)	0	**1**	**1**	**1**	**2**	**2**
(1,3)	**1**	**1**	**1**	0	**1**	**1**

En este ejemplo, se observa como el enlace de salida $(1,2)$ contribuye con dos componentes a los flujos de los enlaces de llegada $(9,10)$ y $(9,11)$. Esto conlleva que estos dos flujos se anulen y solo se tome el flujo proveniente del enlace $(1,3)$. Esta doble contribución de $(1,2)$, se observa en el grafo de la Figura 5.14, donde hay dos caminos que llegan del nodo $(1,2)$ a los nodos $(9,10)$ y $(9,11)$.

5.7.3.2 Restricciones del sistema

De acuerdo con el grafo de la sesión multicast representada en el grafo de la Figura 5.13, se muestra el siguiente conjunto de restricciones derivadas de la aplicación del modelo.

1. Restricciones en los flujos de enlaces de salida:

   Caso I: Flujos con paquetes simples:

$$f_{1\_2} = 1\mathbb{V}2$$
$$f_{1\_3} = 1\mathbb{V}2$$

   Caso II: Flujos con paquetes combinados:

$$f_{1\_2} \neq 0$$
$$f_{1\_3} \neq 0$$

2. Restricciones en los flujos de enlaces de llegada:

   Con el apoyo de la matriz de enlaces reducida Q_r, se tiene:

$$f_{3\_6} = f_{1\_2} \qquad f_{5\_10} = f_{1\_2} + f_{1\_3} \qquad f_{7\_11} = f_{1\_2}$$
$$f_{5\_6} = f_{1\_2} + f_{1\_3} \qquad f_{9\_10} = f_{1\_2} + f_{1\_2} + f_{1\_3} \qquad f_{9\_11} = f_{1\_2} + f_{1\_2} + f_{1\_3}$$

3. Restricciones en los enlaces de llegada y en los nodos sumideros:

   a. Flujos de llegada no nulos:

$$f_{3\_6} \neq 0 \qquad f_{5\_10} \neq 0 \qquad f_{7\_11} \neq 0$$
$$f_{5\_6} \neq 0 \qquad f_{9\_10} \neq 0 \qquad f_{9\_11} \neq 0$$

   b. Combinación lineal de los flujos de llegada a un nodo sumidero no nula:

$$f_{3\_6} + f_{5\_6} \neq 0$$
$$f_{5\_10} + f_{9\_10} \neq 0$$
$$f_{7\_11} + f_{9\_11} \neq 0$$

   c. Matriz de códigos de etiquetas con tamaño $r = 2$:
   Los códigos de este espacio son:

$$\wp = \mathbb{F}_2^2 \backslash \{0\} = \{01,10,11\}$$

   A continuación, en la Tabla 5.30, se presentan las matrices de códigos y la restricción de linealidad independiente entre los flujos entrantes a cada nodo sumidero en T:

Tabla 5.30 Matrices de Código y Restricciones Linealmente Independientes

Para $t = 6$	**Para $t = 10$**	**Para $t = 11$**
$P_6 = \begin{bmatrix} f_{3\_6}^1 & f_{3\_6}^2 \\ f_{5\_6}^1 & f_{5\_6}^2 \end{bmatrix}$	$P_{10} = \begin{bmatrix} f_{5\_10}^1 & f_{5\_10}^2 \\ f_{9\_10}^1 & f_{9\_10}^2 \end{bmatrix}$	$P_{11} = \begin{bmatrix} f_{7\_11}^1 & f_{7\_11}^2 \\ f_{9\_11}^1 & f_{9\_11}^2 \end{bmatrix}$
$f_{3\_6}^1 * f_{5\_6}^2 + f_{3\_6}^2 * f_{5\_6}^1 \neq 0$	$f_{5\_10}^1 * f_{9\_10}^2 + f_{5\_10}^2 * f_{9\_10}^1 \neq 0$	$f_{7\_11}^1 * f_{9\_11}^2 + f_{7\_11}^2 * f_{9\_11}^1 \neq 0$

5.7.3.3 Solución al sistema

Se logran las siguientes respuestas para cada valor de flujo (paquete) en los enlaces de salida y cada valor de flujo (paquete) en los enlaces de llegada:

Caso I: Flujos con paquetes simples:
Códigos de paquetes emergentes por enlaces de salida:

$$f_{1\_2} = 1 = A$$
$$f_{1\_3} = 2 = B$$

Códigos de paquetes entrantes por los enlaces de llegada en la Tabla 5.31:

Tabla 5.31 Códigos de Paquetes Entrantes por los Enlaces de Llegada - Caso I

Para $t = 6$	**Para $t = 10$**	**Para $t = 11$**
$f_{3\_6} = 2 = B$	$f_{5\_10} = 3 = A + B$	$f_{7\_11} = 1 = A$
$f_{5\_6} = 3 = A + B$	$f_{9\_10} = 2 = B$	$f_{9\_11} = 2 = B$

El grafo de sesión multicast con la solución, se muestra en la Figura 5.15(a).

Caso II: Flujos con paquetes combinados:
Códigos de paquetes emergentes por enlaces de salida:

$$f_{1\_2} = 3 = A + B$$
$$f_{1\_3} = 2 = B$$

Códigos de paquetes entrantes por los enlaces de llegada en la Tabla 5.32:

Tabla 5.32 Códigos de Paquetes Entrantes por los Enlaces de Llegada - Caso II

Para $t = 6$	**Para $t = 10$**	**Para $t = 11$**
$f_{3\_6} = 2 = B$	$f_{5\_10} = 1 = A$	$f_{7\_11} = 3 = A + B$
$f_{5\_6} = 1 = A$	$f_{9\_10} = 2 = B$	$f_{9\_11} = 2 = B$

El grafo de sesión multicast con la solución se muestra en la Figura 5.15(b).

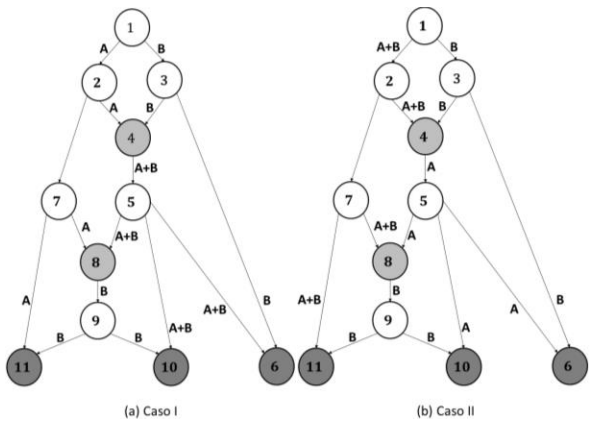


Figura 5.15 Solución en grafo G' de 11 nodos

5.7.4 Ejemplo 4: Sesión multicast de 10 nodos

5.7.4.1 Grafos y Matrices del Sistema

La Figura 5.16 y la Tabla 5.33 muestran el grafo y la matriz de capacidades, respectivamente, de una sesión multicast de 10 nodos, con un nodo fuente ($s = 1$), cinco sumideros ($T = \{6, 7,8,9,10\}$) y flujo máximo $r = 2$ paquetes en $\mathbb{F}_2^l$. Por tener un flujo máximo de 2, los códigos que etiquetarán los paquetes estarán en el espacio $\mathbb{F}_2^2 \backslash \{0\}$.

La Figura 5.16 expone dos situaciones importantes: La primera es que hay dos *enlace de salida-llegada*, el (1,6) y el (1,9) y, la segunda es que los nodos sumideros 6 y 8 también son dos codificadores que reenvían la información saliente a los nodos sumideros 7 y 10, respectivamente.

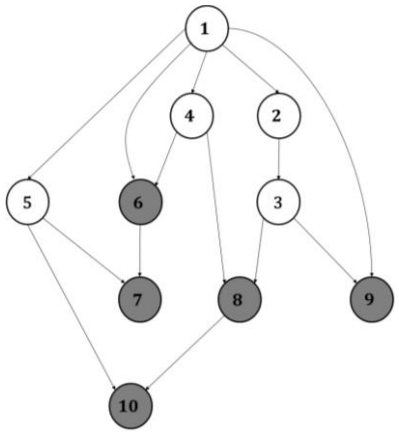


Figura 5.16 Sesión Multicast de 10 nodos

Tabla 5.33 Matriz de Adyacencia (Capacidad) C del Grafo G' de 10 nodos

nodos	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**
1	0	**1**	0	**1**	**1**	**1**	0	0	**1**	0
2	0	0	**1**	0	0	0	0	0	0	0
3	0	0	0	0	0	0	0	**1**	**1**	0
4	0	0	0	0	0	**1**	0	**1**	0	0
5	0	0	0	0	0	0	**1**	0	0	**1**
6	0	0	0	0	0	0	**1**	0	0	0
7	0	0	0	0	0	0	0	0	0	0
8	0	0	0	0	0	0	0	0	0	**1**
9	0	0	0	0	0	0	0	0	0	0
10	0	0	0	0	0	0	0	0	0	0

La Tabla 5.34 muestra los enlaces y sus tipos. De esta tabla también se deduce que los conjuntos de enlaces de salida, enlaces de llegada y enlaces salida-llegada están constituidos por:

$$S = \{(1,2), (1,4), (1,5)\}$$
$$K = \{(1,6), (1,9)\}$$
$$L = \{(3,8), (3,9), (4,6), (4,8), (5,7), (5,10), (6,7), (8,10)\}$$

Tabla 5.34 Tabla de Enlaces

Índice	1	2	3	4	5	6	7	8	9	10	11	12	13	14
Origen	1	1	1	1	1	2	3	3	4	4	5	5	6	8
Destino	2	4	5	6	9	3	8	9	6	8	7	10	7	10
Tipo	1	1	1	3	3	0	2	2	2	2	2	2	2	2

El grafo lineal dirigido 𝔊 se presenta en la Figura 5.17:

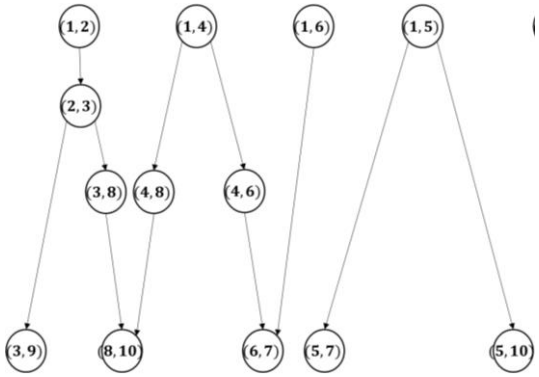


Figura 5.17 Grafo Lineal Dirigido para 11 nodos

Se observa que el camino más largo es de cinco saltos, por lo cual solo se puede iterar hasta $n = 3$ y en $n = 4, F^4 = 0$. En la Tabla 5.35 hasta la Tabla 5.37, se muestran las iteraciones desde un salto hasta alcanzar los tres saltos en F^3. La Tabla 5.38 muestra la matriz de enlaces y la Tabla 5.39 muestra la matriz de enlaces reducida.

Tabla 5.35 Matriz de un salto *F*

Enlace	**(1,2)**	**(1,4)**	**(1,5)**	**(1,6)**	**(1,9)**	**(2,3)**	**(3,8)**	**(3,9)**	**(4,6)**	**(4,8)**	**(5,7)**	**(5,10)**	**(6,7)**	**(8,10)**
(1,2)	0	0	0	0	0	**1**	0	0	0	0	0	0	0	0
(1,4)	0	0	0	0	0	0	0	0	**1**	**1**	0	0	0	0
(1,5)	0	0	0	0	0	0	0	0	0	0	**1**	**1**	0	0
(1,6)	0	0	0	0	0	0	0	0	0	0	0	0	**1**	0
(1,9)	0	0	0	0	0	0	0	0	0	0	0	0	0	0
(2,3)	0	0	0	0	0	0	**1**	**1**	0	0	0	0	0	0
(3,8)	0	0	0	0	0	0	0	0	0	0	0	0	0	**1**
(3,9)	0	0	0	0	0	0	0	0	0	0	0	0	0	0
(4,6)	0	0	0	0	0	0	0	0	0	0	0	0	**1**	0
(4,8)	0	0	0	0	0	0	0	0	0	0	0	0	0	**1**
(5,7)	0	0	0	0	0	0	0	0	0	0	0	0	0	0
(5,10)	0	0	0	0	0	0	0	0	0	0	0	0	0	0
(6,7)	0	0	0	0	0	0	0	0	0	0	0	0	0	0
(8,10)	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Tabla 5.36 Matriz de dos saltos F^2

Enlace	**(1,2)**	**(1,4)**	**(1,5)**	**(1,6)**	**(1,9)**	**(2,3)**	**(3,8)**	**(3,9)**	**(4,6)**	**(4,8)**	**(5,7)**	**(5,10)**	**(6,7)**	**(8,10)**
(1,2)	0	0	0	0	0	0	**1**	**1**	0	0	0	0	0	0
(1,4)	0	0	0	0	0	0	0	0	0	0	0	0	**1**	**1**
(1,5)	0	0	0	0	0	0	0	0	0	0	0	0	0	0
(1,6)	0	0	0	0	0	0	0	0	0	0	0	0	0	0
(1,9)	0	0	0	0	0	0	0	0	0	0	0	0	0	0
(2,3)	0	0	0	0	0	0	0	0	0	0	0	0	0	**1**
(3,8)	0	0	0	0	0	0	0	0	0	0	0	0	0	0
(3,9)	0	0	0	0	0	0	0	0	0	0	0	0	0	0
(4,6)	0	0	0	0	0	0	0	0	0	0	0	0	0	0
(4,8)	0	0	0	0	0	0	0	0	0	0	0	0	0	0
(5,7)	0	0	0	0	0	0	0	0	0	0	0	0	0	0
(5,10)	0	0	0	0	0	0	0	0	0	0	0	0	0	0
(6,7)	0	0	0	0	0	0	0	0	0	0	0	0	0	0
(8,10)	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Tabla 5.37 Matriz de tres saltos F^3

Enlace	(1,2)	(1,4)	(1,5)	(1,6)	(1,9)	(2,3)	(3,8)	(3,9)	(4,6)	(4,8)	(5,7)	(5,10)	(6,7)	(8,10)
(1,2)	0	0	0	0	0	0	0	0	0	0	0	0	0	**1**
(1,4)	0	0	0	0	0	0	0	0	0	0	0	0	0	0
(1,5)	0	0	0	0	0	0	0	0	0	0	0	0	0	0
(1,6)	0	0	0	0	0	0	0	0	0	0	0	0	0	0
(1,9)	0	0	0	0	0	0	0	0	0	0	0	0	0	0
(2,3)	0	0	0	0	0	0	0	0	0	0	0	0	0	0
(3,8)	0	0	0	0	0	0	0	0	0	0	0	0	0	0
(3,9)	0	0	0	0	0	0	0	0	0	0	0	0	0	0
(4,6)	0	0	0	0	0	0	0	0	0	0	0	0	0	0
(4,8)	0	0	0	0	0	0	0	0	0	0	0	0	0	0
(5,7)	0	0	0	0	0	0	0	0	0	0	0	0	0	0
(5,10)	0	0	0	0	0	0	0	0	0	0	0	0	0	0
(6,7)	0	0	0	0	0	0	0	0	0	0	0	0	0	0
(8,10)	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Tabla 5.38 Matriz de Enlaces Q

Enlace	(1,2)	(1,4)	(1,5)	(1,6)	(1,9)	(2,3)	(3,8)	(3,9)	(4,6)	(4,8)	(5,7)	(5,10)	(6,7)	(8,10)
(1,2)	0	0	0	0	0	**1**	**1**	**1**	0	0	0	0	0	**1**
(1,4)	0	0	0	0	0	0	0	0	**1**	**1**	0	0	**1**	**1**
(1,5)	0	0	0	0	0	0	0	0	0	0	**1**	**1**	0	0
(1,6)	0	0	0	0	0	0	0	0	0	0	0	0	**1**	0
(1,9)	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Tabla 5.39 Matriz de Enlaces reducida Q_r

Enlace	(1,6)	(1,9)	(3,8)	(3,9)	(4,6)	(4,8)	(5,7)	(5,10)	(6,7)	(8,10)
(1,2)	**0**	**0**	**1**	**1**	0	0	0	0	0	**1**
(1,4)	**0**	**0**	0	0	**1**	**1**	0	0	**1**	**1**
(1,5)	**0**	**0**	0	0	0	0	**1**	**1**	0	0
(1,6)	**0**	**0**	0	0	0	0	0	0	**1**	0
(1,9)	**0**	**0**	0	0	0	0	0	0	0	0

En Q_r se observa que las columnas (1,6) y (1,9) están en ceros, a raíz que estas corresponden a enlaces *salida-llegada*, los cuales permiten una comunicación directa del nodo fuente a los nodos sumideros 6 y 9.

5.7.4.2 Ecuaciones del sistema

De acuerdo con el grafo de la sesión multicast representada en el grafo de la Figura 5.16, se muestra el siguiente conjunto de restricciones derivadas de la aplicación del modelo.

1. Restricciones en los flujos de enlaces de salida:

   Caso I: Flujos con paquetes simples:

   $f_{1\_2} = 1 \vee 2$ $\quad f_{1\_5} = 1 \vee 2$ $\quad f_{1\_9} = 1 \vee 2$
   $f_{1\_4} = 1 \vee 2$ $\quad f_{1\_6} = 1 \vee 2$

   Caso II: Flujos con paquetes combinados:

   $f_{1\_2} \neq 0$ $\quad f_{1\_5} \neq 0$ $\quad f_{1\_9} \neq 0$
   $f_{1\_4} \neq 0$ $\quad f_{1\_6} \neq 0$

2. Restricciones en los flujos de enlaces de llegada:

   Con el apoyo de la matriz de enlaces reducida Q_r, se tiene:

$$f_{4\_6} = f_{1\_4} \qquad f_{5\_7} = f_{1\_5} \qquad f_{3\_8} = f_{1\_2} \qquad f_{3\_9} = f_{1\_2} \qquad f_{5\_10} = f_{1\_5}$$
$$f_{6\_7} = f_{1\_4} + f_{1\_6} \qquad f_{4\_8} = f_{1\_4} \qquad f_{8\_10} = f_{1\_2} + f_{1\_4}$$

3. Restricciones en los enlaces de llegada y en los nodos sumideros:

   a. Flujos de llegada no nulos:

$$f_{1\_6} \neq 0 \qquad f_{5\_7} \neq 0 \qquad f_{3\_8} \neq 0 \qquad f_{1\_9} \neq 0 \qquad f_{5\_10} \neq 0$$
$$f_{4\_6} \neq 0 \qquad f_{6\_7} \neq 0 \qquad f_{4\_8} \neq 0 \qquad f_{3\_9} \neq 0 \qquad f_{8\_10} \neq 0$$

   b. Combinación lineal de los flujos de llegada a un nodo sumidero no nula:

$$f_{1\_6} + f_{4\_6} \neq 0$$
$$f_{5\_7} + f_{6\_7} \neq 0$$
$$f_{3\_8} + f_{4\_8} \neq 0$$
$$f_{1\_9} + f_{3\_9} \neq 0$$
$$f_{5\_10} + f_{8\_10} \neq 0$$

   c. Matriz de códigos de etiquetas con tamaño $r = 2$:
   Los códigos de este espacio son:

$$\wp = \mathbb{F}_2^2 \backslash \{0\} = \{01,10,11\}$$

   A continuación se presentan las matrices de códigos y la restricción de linealidad independiente entre los flujos entrantes a cada nodo sumidero en T:
   Para $t = 6$:

$$P_6 = \begin{bmatrix} f_{1\_6}^1 & f_{1\_6}^2 \\ f_{4\_6}^1 & f_{4\_6}^2 \end{bmatrix}$$

$$f_{1\_6}^1 * f_{4\_6}^2 + f_{1\_6}^2 * f_{4\_6}^1 \neq 0$$

   Para $t = 7$:

$$P_7 = \begin{bmatrix} f_{5\_7}^1 & f_{5\_7}^2 \\ f_{6\_7}^1 & f_{6\_7}^2 \end{bmatrix}$$

$$f_{5\_7}^1 * f_{6\_7}^2 + f_{5\_7}^2 * f_{6\_7}^1 \neq 0$$

   Para $t = 8$:

$$P_8 = \begin{bmatrix} f_{3\_8}^1 & f_{3\_8}^2 \\ f_{4\_8}^1 & f_{4\_8}^2 \end{bmatrix}$$

$$f_{3\_8}^1 * f_{4\_8}^2 + f_{3\_8}^2 * f_{4\_8}^1 \neq 0$$

Para $t = 9$:

$$P_9 = \begin{bmatrix} f^1_{1\_9} & f^2_{1\_9} \\ f^1_{3\_9} & f^2_{3\_9} \end{bmatrix}$$

$$f^1_{1\_9} * f^2_{3\_9} + f^2_{1\_9} * f^1_{3\_9} \neq 0$$

Para $t = 10$:

$$P_{10} = \begin{bmatrix} f^1_{5\_10} & f^2_{5\_10} \\ f^1_{8\_10} & f^2_{8\_10} \end{bmatrix}$$

$$f^1_{5\_10} * f^2_{8\_10} + f^2_{5\_10} * f^1_{8\_10} \neq 0$$

5.7.4.3 Solución al sistema

Se logran las siguientes respuestas para cada valor de flujo (paquete) en los enlaces de salida y cada valor de flujo (paquete) en los enlaces de llegada:

Caso I: Flujos con paquetes simples:
Códigos de paquetes emergentes por enlaces de salida:

$$f_{1\_2} = 2 = B$$
$$f_{1\_4} = 1 = A$$
$$f_{1\_5} = 2 = B$$
$$f_{1\_6} = 2 = B$$
$$f_{1\_9} = 1 = A$$

Códigos de paquetes entrantes por los enlaces de llegada en la Tabla 5.40:

Tabla 5.40 Códigos de Paquetes Entrantes por los Enlaces de Llegada - Caso I

Para $t = 6$	**Para $t = 7$**	**Para $t = 8$**	**Para $t = 9$**	**Para $t = 10$**
$f_{1\_6} = 2 = B$	$f_{5\_7} = 2 = B$	$f_{3\_8} = 2 = B$	$f_{1\_9} = 1 = A$	$f_{5\_10} = 2 = B$
$f_{4\_6} = 1 = A$	$f_{6\_7} = 3 = A + B$	$f_{4\_8} = 1 = A$	$f_{3\_9} = 2 = B$	$f_{8\_10} = 3 = A + B$

El grafo de sesión multicast con la solución, se muestra en la Figura 5.18(a).

Caso II: Flujos con paquetes combinados:
Códigos de paquetes emergentes por enlaces de salida:

$$f_{1\_2} = 3 = A + B$$
$$f_{1\_4} = 1 = A$$
$$f_{1\_5} = 3 = A + B$$
$$f_{1\_6} = 3 = A + B$$
$$f_{1\_9} = 1 = A$$

Códigos de paquetes entrantes por los enlaces de llegada en la Tabla 5.41:

Tabla 5.41 Códigos de Paquetes Entrantes por los Enlaces de Llegada - Caso II

Para $t = 6$	**Para $t = 7$**	**Para $t = 8$**	**Para $t = 9$**	**Para $t = 10$**
$f_{1\_6} = 3 = A + B$ $f_{4\_6} = 1 = A$	$f_{5\_7} = 3 = A + B$ $f_{6\_7} = 2 = B$	$f_{3\_8} = 3 = A + B$ $f_{4\_8} = 1 = A$	$f_{1\_9} = 1 = A$ $f_{3\_9} = 3 = A + B$	$f_{5\_10} = 3 = A + B$ $f_{8\_10} = 2 = B$

El grafo de sesión multicast con la solución se muestra en la Figura 5.18(b).

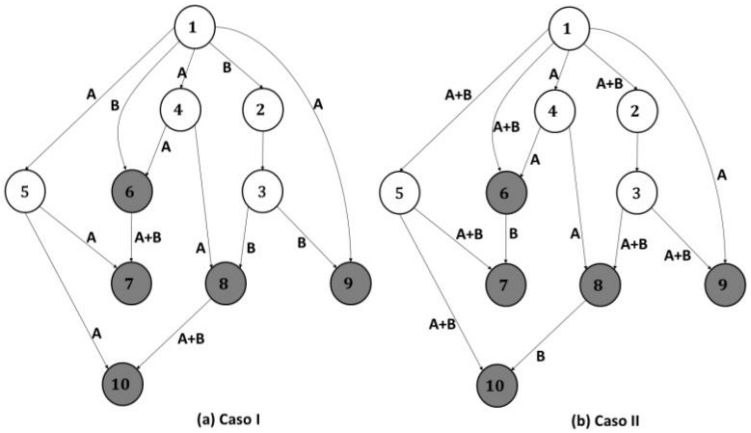


Figura 5.18 Solución en grafo G'

5.7.5 Ejemplos adicionales

5.7.5.1 Sesión multicast de 7 nodos sin codificadores

En el grafo de sesión multicast, representado en la Figura 5.19(a), no hay nodos codificadores, consta de 3 sumideros y $r = 2$. La solución existente corresponde solo al Caso II, y la del Caso I no se logra, dado que no es posible enviar paquetes simples por los tres enlaces de salida y que se garantice la recepción de los paquetes generados por el computador emisor en los nodos sumideros.

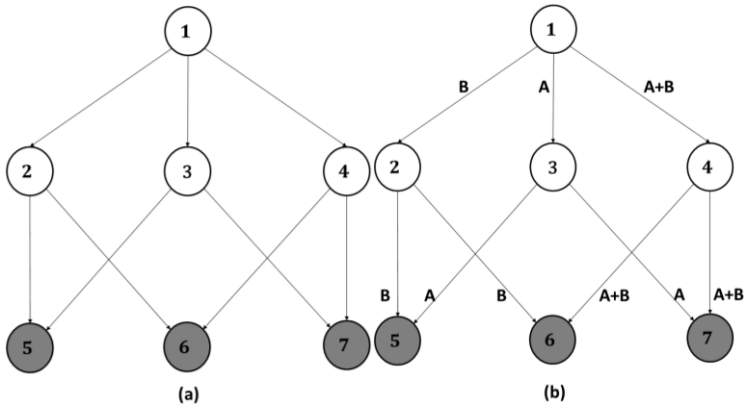


Figura 5.19 (a) Sesión Multicast de 7 nodos sin codificadores. (b) Solución solo con paquetes combinados

5.7.5.2 Sesión multicast de 11 nodos sin codificadores y sin solución

El grafo de sesión multicast mostrado en la Figura 5.20, tomado de [118], no tiene codificadores, consta de 6 sumideros y $r = 2$. Se observa que en este grafo no existe solución para ninguno de los

dos casos. En [118] proponen un aumento de la capacidad a 2 paquetes, mientras mantienen el tamaño del buffer, es decir, el tamaño es de un paquete. Esto no concordaría con el modelo propuesto, donde se mantiene la misma capacidad de un paquete en cada enlace.

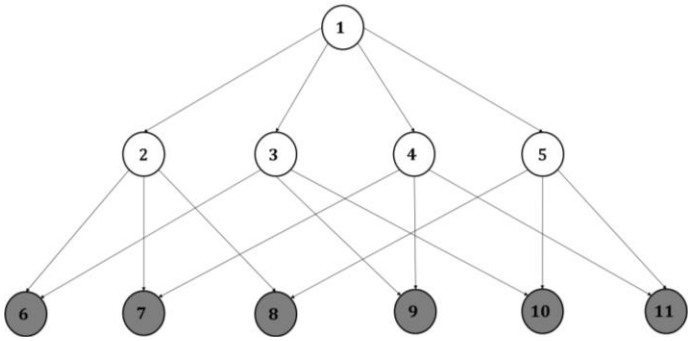


Figura 5.20 Sesión Multicast de 11 nodos sin solución

5.8 Resumen de características del método

5.8.1 Características del método

1. Si el flujo máximo es r para una sesión multicast con un solo nodo fuente, se enviarán, desde este nodo, r paquetes de longitud l símbolos.
2. Los símbolos son considerados bits y, por ende, pertenecen al campo $\mathbb{F}_2$.
3. A cada paquete de longitud l se le considera una unidad de información que se transmitirá por los enlaces de la sesión multicast hasta alcanzar cada uno de los nodos sumideros en T.
4. Los r paquetes que se generan en el nodo fuente s, tienen longitud l y constituyen la unidad de medida de información, no importando cuál sea el tamaño de l. Todos los paquetes en la red (sesión multicast) tienen el mismo tamaño l.
5. Los r paquetes que se pueden generar y enviar a partir del nodo fuente se etiquetan con un código binario (manipulado como un número entero que está en $\mathbb{N}$) de longitud r que pertenece al espacio vectorial $\mathbb{F}_2^r$ y corresponden con el $i\text{-}ésimo$ vector de la base canónica de $\mathbb{F}_2^r$. $u_i = \left[0^{i-1}, 1, 0^{r-i}\right], i = 1, \dots, r$, donde 0^j significa un vector fila de ceros de longitud j. Las etiquetas de los paquetes son linealmente independientes.
6. Otra forma más simple de nombrar los paquetes es a través de las letras del alfabeto latino, donde cada letra en orden topológico representa un vector de la base canónica de $\mathbb{F}_2^r$ en orden creciente. Es así como en la Tabla 5.42, se puede establecer la relación para $r = 4$:

Tabla 5.42 Ejemplo de relación de mnemónicos, vectores y enteros

Letra del alfabeto	Vector base canónica de $\mathbb{F}_2^r$	Valor entero
A	$u_1 = [0001]$	1
B	$u_2 = [0010]$	2
C	$u_3 = [0100]$	4
D	$u_4 = [1000]$	8

7. El ancho de banda de cada enlace en la red es de tamaño l símbolos por unidad de tiempo (l bits por unidad de tiempo), es decir, la capacidad es de una unidad de información por unidad de tiempo.
8. En cada nodo de codificación/enrutamiento se establece una combinación lineal en $\mathbb{F}_2$ entre los bits correspondientes a la misma posición en los paquetes (vectores) que ingresan al nodo. Lógicamente se utilizan como coeficientes de codificación los símbolos (bits) en $\mathbb{F}_2$. Es importante resaltar, que los nodos de enrutamiento solo reenvían los paquetes que a ellos ingresan; así, que en estos, el proceso de combinación lineal no existe.
9. El algoritmo, que resuelve las ecuaciones con Z3, debe probar todas las posibles combinaciones hasta encontrar una solución correcta. En el peor de los casos, alcanzará las cotas máximas definidas en (5.4) para paquetes simples y en (5.5) para paquetes combinados, si la solución está en el último lugar del conjunto de combinaciones posibles o no existen soluciones para el sistema lineal de ecuaciones formado a partir del grafo multicast de mínimo flujo máximo.

5.8.2 Diferencias con el algoritmo polinomial de Jaggi *et al.* [85]

1. El algoritmo polinomial se basa en encontrar los coeficientes de codificación global en el espacio $\mathbb{F}_2^r$, en cada nodo $o(e)$ donde inicia un enlace e, de tal forma que todos los sumideros puedan reconstruir la información original enviada por el nodo fuente a partir de los símbolos recibidos.
2. El algoritmo propuesto no se basa, prioritariamente, en encontrar los coeficientes de codificación global, sino que busca el orden en que se deben arreglar los r-paquetes entre los enlaces de salida, de tal forma que se pueda obtener la información original a partir de los paquetes (simples y combinaciones lineales de los mismos) que arriben por los enlaces de llegada a los nodos sumideros.
3. El algoritmo polinomial introduce r-enlaces paralelos virtuales desde un nuevo nodo s' hasta s, los cuales transportan los símbolos (o paquetes) entrantes hacia el nodo fuente s.
4. El algoritmo propuesto no define enlaces auxiliares, sino que trabaja inmediatamente con las etiquetas de los r paquetes formados por l-bits. Cada paquete $p_k \in \mathbb{F}_2^l$, donde $1 \leq k \leq r$.
5. El algoritmo polinomial utiliza los vectores de la base canónica de $\mathbb{F}_2^r$, $u_i = [0^{i-1}, 1, 0^{r-i}]$ para iniciar los vectores de codificación global y etiquetar los enlaces virtuales agregados.
6. El algoritmo propuesto utiliza los vectores de la base canónica de $\mathbb{F}_2^r$ para representar de manera única la identidad de cada uno de los r paquetes que se generarán en el nodo fuente s. Estos vectores representan la etiqueta de cada paquete, la cual se puede mostrar como una letra de un alfabeto.
7. El algoritmo polinomial y otros (sección 2.4.3), considera el tamaño del campo al que pertenecen los símbolos que se van a transmitir desde el nodo fuente en una sesión multicast, como una restricción para solucionar el problema de LNC. Para el polinomial la restricción es $|\mathbb{F}| \geq |T|$.
8. Sin embargo, el algoritmo propuesto solo se preocupa por conocer el flujo máximo r (es decir, r paquetes distintos desde el nodo fuente) en la sesión multicast, y a partir de este define que por cada enlace de salida, se debe generar un vector de la base canónica de $\mathbb{F}_2^r$ de los r-posibles (o una combinación de estos), con la posibilidad de que se repitan entre enlaces de salida. Es decir, por más de un enlace, puede originar el mismo vector para paquetes simples (o la letra

con la que corresponda) o la combinación de vectores (o la combinación lineal de letras a la que correspondan, si la solución es con paquetes combinados).

Capítulo 6

Pruebas y Resultados

Las pruebas a los métodos expuestos en los capítulos 2 al 4, se desarrollaron con 37 grafos de comunicaciones. A continuación se expondrán las características de los grafos, los resultados de las pruebas según el número de soluciones posibles utilizando Network Coding y el rendimiento de los algoritmos según el tiempo de cómputo para reducir el grafo de mínimo flujo máximo a una red multicast unisesión. Es importante resaltar que de estos 37 grafos, dos no logran tener solución dentro de un modelo multicast donde se aplique Network Coding, y en uno la red termina teniendo un flujo máximo de uno, el cual también se descarta.

6.1 Caracterización de los grafos de comunicaciones G para prueba

Los grafos de comunicaciones de prueba se caracterizan por las siguientes variables:

Número de nodos del grafo de comunicaciones: El número de nodos de los grafos está comprendido en un rango de 7 a 54 nodos. Este valor corresponde a $|V|$ para cada grafo.

Número de nodos sumideros: Los grafos de prueba constan de nodos sumideros en el rango de 2 a 7. Este valor corresponde a $|T|$ para cada grafo.

Número de enlaces: Es el número de enlaces que conectan los nodos que constituyen el grafo. Este valor corresponde a $|E|$ para cada grafo.

Flujos máximos de los grafos: Los flujos máximos obtenidos en estos grafos están comprendidos entre 1 y 6. Este valor corresponde al valor r para cada grafo.

El número de grafos correspondiente por cada cantidad de nodos, sumideros y flujos se observa en la Tabla 6.1. La Tabla 6.2 muestra la descripción de cada grafo de prueba.

Tabla 6.1 Número de grafos de comunicaciones por cantidad de (a) nodos, (b) sumideros (c) flujos

Número de Nodos	Número de Grafos
7	3
9	1
10	1
11	7
12	3
13	4
15	1
17	1
18	2
20	4
23	1

Número de Sumideros	Número de Grafos
2	4
3	16
4	5
5	3
6	6
7	3

(b)

Número de Flujos	Número de Grafos
1	1
2	22

27	1
31	1
35	1
38	2
45	1
46	1
47	1
54	1

(a)

3	5
4	4
5	1
6	4

(c)

Tabla 6.2 Descripción de cada grafo G

Sec	Archivo-Grafo	Nodos	Sumideros	Enlaces	Flujos
1	7grafo2fm1v2des	7	2	9	2
2	7grafo2fm2v3des	7	3	9	2
3	7grafo2fm3v3des	7	3	13	2
4	9grafo2fm1v3des	9	3	13	2
5	10grafo2fm1v3des	10	3	16	2
6	11grafo2fm1v2des	11	2	15	2
7	11grafo2fm1v3des	11	3	15	2
8	11grafo2fm1v6des	11	6	16	2
9	11grafo2fm2v3des	11	3	15	2
10	11grafo2fm3v3des	11	3	18	2
11	11grafo2fm4v3des	11	3	18	2
12	11grafo4fm1v2des	11	2	22	4
13	12grafo2fm1v3des	12	3	21	2
14	12grafo2fm2v3des	12	3	26	2
15	12grafo3fm1v3des	12	3	24	3
16	13grafo1fm2v3des	13	3	22	1
17	13grafo2fm1v5des	13	5	27	2
18	13grafo3fm1v4des	13	4	27	3
19	13grafo3fm2v4des	13	4	27	3
20	15grafo2fm1v4des	15	4	21	2
21	17grafo2fm1v6des	17	6	26	2
22	18grafo2fm1v6des	18	6	36	2
23	18grafo2fm2v6des	18	6	36	2
24	20grafo2fm1v4des	20	4	50	2
25	20grafo4fm1v3des	20	3	47	4
26	20grafo4fm2v3des	20	3	48	4
27	20grafo5fm1v3des	20	3	49	5
28	23grafo6fm1v2des	23	2	50	6
29	27grafo4fm1v5des	27	5	77	4
30	31grafo2fm1v7des	31	7	76	2
31	35grafo2fm1v6des	35	6	87	2
32	38grafo2fm1v6des	38	6	81	2
33	38grafo3fm1v7des	38	7	101	3
34	45grafo6fm1v3des	45	3	141	6
35	46grafo6fm1v4des	46	4	148	6
36	47grafo6fm1v5des	47	5	155	6
37	54grafo3fm1v7des	54	7	142	3

Los grafos de comunicaciones G están almacenados individualmente en archivos de texto que contienen el listado de los nodos sumideros y a continuación la matriz de adyacencia correspondiente al grafo. Se asume, en todos los casos, que el índice uno corresponde al nodo fuente.

Los 37 grafos de comunicaciones propuestos son considerados para llevar a cabo las pruebas de reducción al grafo multicast unisesión de mínimo flujo máximo con los cinco métodos planteados. Sin embargo, solo se tienen en cuenta 34 grafos multicast unisesión resultantes para llevar a cabo las pruebas al algoritmo de solución propuesto en el Capítulo 5, ya que dos de éstos no tienen solución para el sistema de ecuaciones planteado sobre Network Coding, y el tercero tiene un flujo máximo de uno.

Las soluciones de los algoritmos, se implementaron en el lenguaje de programación Matlab [119] en razón que es un lenguaje orientado al manejo de arreglos, lo cual facilitó la programación de los grafos y conjuntos. Además, contiene la caja de herramientas (*toolbox*) *biograph* que presenta facilidades para la construcción de los grafos.

Los nombres de los archivos de prueba tienen la siguiente notación: *nn*grafo*ff*m*vv*d*d*des.txt (Tabla 6.2), donde:

nn: Entero correspondiente al número de nodos del grafo de comunicaciones G.

f: Entero correspondiente al mínimo flujo máximo del grafo multicast unisesión G'.

v: Entero correspondiente al número de versión del grafo. Esto implica que el grafo que representa es similar a otro donde se pudo haber eliminado un enlace o un nodo.

dd: Entero que representa al número de nodos sumideros específicos para la comunicación multicast.

6.2 Valoración de los métodos de solución

Los métodos de reducción del grafo general de comunicaciones al grafo multicast unisesión de mínimo flujo máximo se valorarán según los siguientes criterios:

6.2.1 Número de soluciones válidas

De acuerdo con el número de grafos de comunicaciones de pruebas, un método será mejor que otro, si halla el mayor número de grafos multicast unisesión, con soluciones al sistema lineal de ecuaciones soportados por Network Coding, que se configure en los nodos sumideros. En la medida en que las soluciones se deriven de un escenario donde se envíen paquetes simples desde el nodo fuente, en vez de paquetes combinados, el método estará mejor posicionado.

6.2.2 Número de nodos codificadores

El número de nodos codificadores es relevante al momento de determinar cuál solución es mejor. En los métodos de enrutamiento multicast clásicos, los nodos enrutadores solo actúan con la política de almacenamiento y reenvío; sin embargo con la aplicación de Network Coding, en los nodos codificadores el trabajo de procesamiento aumenta, ya que deben realizar la operación de

codificación de los paquetes entrantes, además de las dos tareas tradicionales. En consecuencia, el interés es que en una red multicast unisesión de mínimo flujo máximo, haya el menor número de codificadores para incrementar el rendimiento en el proceso de entrega de los paquetes en su destino final.

6.2.3 Soluciones Conjuntas

Al comparar los cinco métodos de reducción sobre un grafo de comunicaciones específico para obtener un grafo multicast unisesión de mínimo flujo máximo, se considera como el mejor, el (los) que logre(n) la solución con paquetes simples, antes que con paquetes combinados y en segunda instancia contenga(n) menos nodos codificadores. Más de un método puede tener la mejor solución; es decir, una misma solución puede ser resultado de la reducción llevada a cabo por más de un método; por tanto, si una mejor solución es obtenida por más de un método, se contabiliza una vez para cada uno de ellos. En síntesis, la medida de soluciones conjuntas puede ser mínimo uno y máximo cinco.

6.2.4 Soluciones Individuales

Esta valoración consiste en que existe un método que fue el único que pudo obtener la mejor solución del G', dentro de la comparación de soluciones respecto a un grafo de comunicaciones G. En el caso en que una mejor solución sea obtenida por más de un método, no se tiene en cuenta (se contabiliza en la variable de soluciones conjuntas). Es importante destacar, que el conjunto de soluciones individuales hace parte de las soluciones conjuntas.

6.2.5 Tiempo de cómputo

Los métodos también se miden por el tiempo de cómputo utilizado durante su ejecución para alcanzar el grafo multicast unisesión de mínimo flujo máximo. Los tiempos de ejecución están determinados por el número de nodos del grafo, el número de nodos sumideros y el mínimo flujo máximo del grafo multicast.

6.3 Resultados de las pruebas según número de soluciones y número de codificadores

La expectativa de los métodos planteados, es que a partir de un grafo general de comunicaciones G, en los que están definidos exactamente el nodo de fuente s y el conjunto de nodos sumideros T, se pueda reducir al grafo multicast unisesión de mínimo flujo máximo G', que permita el envío simultáneo de r paquetes desde el nodo fuente hacia los nodos sumideros.

La Tabla 6.3 muestra los resultados de la ejecución de los cinco métodos con cada uno de los grafos de prueba. En esta tabla se etiquetan las columnas con los nombres de los cinco métodos así: Disyuntos o Disy (Rutas Disyuntas), BFS_PRI o B_PR (BFS con eliminación del primer camino), BFS_ML o B_ML (BFS con eliminación del camino más largo), DFS_PRI o D_PR (DFS con eliminación del primer camino) y DFS_ML o D_ML (DFS con eliminación del camino más largo).

Tabla 6.3 Resultados de pruebas con los cinco métodos

		Disyuntos			BFS_PRI			BFS_ML			DFS_PRI			DFS_ML			Mejores soluciones conjunta					Mejor solución única				
Sec	**Archivo-Grafo**	**S**	**C**	**Co**	**S**	**C**	**Co**	**S**	**C**	**Co**	**S**	**C**	**Co**	**S**	**C**	**Co**	**Disy**	**B_PR**	**B_ML**	**D_PR**	**D_ML**	**Disy**	**B_PR**	**B_ML**	**D_PR**	**D_ML**
1	7grafo2fm1v2des	x	x	1	x	x	1	x	x	1	x	x	1	x	x	1	x	x	x	x	x					
2	7grafo2fm2v3des		x	0		x	0		x	0		x	0		x	0	x	x	x	x	x					
3	7grafo2fm3v3des		x	0	x	x	1		x	0	x	x	1	x	x	1		x		x	x					
4	9grafo2fm1v3des	x	x	1	x	x	1	x	x	1	x	x	1	x	x	1	x	x	x	x	x					
5	10grafo2fm1v3des	x	x	0		x	0		x	0		x	0		x	0	x					x				
6	11grafo2fm1v2des	x	x	1	x	x	1	x	x	1	x	x	1	x	x	1	x	x	x	x	x					
7	11grafo2fm1v3des	x	x	2	x	x	2	x	x	2	x	x	2	x	x	2	x	x	x	x	x					
8	11grafo2fm1v6des																									
9	11grafo2fm2v3des	x	x	1	x	x	1	x	x	1	x	x	1	x	x	1	x	x	x	x	x					
10	11grafo2fm3v3des	x	x	1			2				x	x	3	x	x	1	x				x					
11	11grafo2fm4v3des	x	x	1							x	x	1	x	x	1	x			x	x					
12	11grafo4fm1v2des	x	x	1	x	x	1	x	x	1	x	x	0	x	x	0				x	x					
13	12grafo2fm1v3des	x	x	1	x	x	0		x	2	x	x	3	x	x	3		x					x			
14	12grafo2fm2v3des	x	x	0	x	x	1	x	x	1	x	x	2	x	x	2	x					x				
15	12grafo3fm1v3des	x	x	0	x	x	0	x	x	0	x	x	0	x	x	0	x	x	x	x	x					
16	13grafo1fm2v3des																									
17	13grafo2fm1v5des	x	x	0	x	x	0	x	x	0	x	x	0	x	x	0	x	x	x	x	x					
18	13grafo3fm1v4des	x	x	0		x	0	x	x	0		x	0	x	x	0	x	x	x	x	x					
19	13grafo3fm2v4des	x	x	0	x	x	0	x	x	0	x	x	0	x	x	0	x	x	x	x	x					
20	15grafo2fm1v4des	x	x	1	x	x	1	x	x	1	x	x	1	x	x	1	x	x	x	x	x					
21	17grafo2fm1v6des																									
22	18grafo2fm1v6des	x	x	0	x	x	2	x	x	1	x	x	1	x	x	1	x					x				
23	18grafo2fm2v6des	x	x	0	x	x	2			1			2			2	x					x				
24	20grafo2fm1v4des	x	x	2		x	2		x	2		x	3		x	3	x					x				
25	20grafo4fm1v3des	x	x	2	x	x	2	x	x	2	x	x	1	x	x	2				x					x	
26	20grafo4fm2v3des	x	x	1	x	x	2	x	x	2	x	x	2	x	x	3	x					x				
27	20grafo5fm1v3des	x	x	2	x	x	3	x	x	3	x	x	3	x	x	3	x					x				
28	23grafo6fm1v2des	x	x	3	x	x	4	x	x	3	x	x	4	x	x	4	x		x							
29	27grafo4fm1v5des	x	x	5			8			7	x	x	8	x	x	8	x					x				
30	31grafo2fm1v7des	x	x	1			2		x	0			10		x	10	x					x				
31	35grafo2fm1v6des	x	x	3	x	x	5		x	2			8			8	x					x				
32	38grafo2fm1v6des	x	x	2		x	2			2			6		x	3	x					x				
33	38grafo3fm1v7des	x	x	8		x	6		x	5		x	14		x	13	x					x				
34	45grafo6fm1v3des	x	x	4	x	x	3	x	x	3	x	x	6	x	x	6		x	x							
35	46grafo6fm1v4des	x	x	8	x	x	8	x	x	8	x	x	9	x	x	9	x	x	x							
36	47grafo6fm1v5des			5	x	x	8	x	x	8			12			12		x	x							
37	54grafo3fm1v7des	x	x	7		x	9	x	x	7			15		x	14	x		x							
	Total	**31**	**33**	**64**	**23**	**30**	**80**	**21**	**29**	**67**	**23**	**28**	**121**	**24**	**31**	**116**	**28**	**16**	**16**	**15**	**15**	**12**	**1**	**0**	**1**	**0**

Las columnas etiquetadas con S y C representan si las soluciones se desarrollaron con paquetes simples o paquetes combinados, respectivamente. La columna etiquetada con Co significa el número de nodos codificadores que arrojó la solución. Al final, se totaliza cada columna por método. Resalta, que de los 37 grafos probados, solo se cuentan 34 al momento de hacer los cálculos, ya que tres (Grafos 8, 16 y 21) fueron excluidos, y las razones explicadas previamente.

Se observa en la Tabla 6.4 y la Figura 6.1 (a) que el sistema lineal de ecuaciones formulado bajo la técnica de Network Coding, tiene el mayor porcentaje de soluciones para los grafos G' que se derivan a partir del método de los caminos disyuntos con respecto a los otros cuatro métodos. El mayor porcentaje de soluciones con este método, se alcanza cuando se utilizan paquetes combinados linealmente en el nodo fuente. De igual forma, se logra un porcentaje significativo de soluciones, con este método, cuando se envían paquetes simples desde el nodo fuente. Es importante señalar que el método DFS, con la eliminación del camino más largo, presenta un porcentaje importante del 91% de soluciones al sistema lineal de ecuaciones combinando los paquetes por las salidas del nodo fuente.

La Tabla 6.4 y la Figura 6.1 (b) muestran que el número de nodos codificadores presenta el más bajo valor promedio cuando se aplica el método de caminos disyuntos para la reducción del grafo de comunicaciones.

Tabla 6.4 Comparación de los métodos de reducción sobre 34 grafos G probados

Método de Reducción	**Paquetes Simples**		**Paquetes Combinados**		**Promedio Codificadores**
	Soluciones	**%**	**Soluciones**	**%**	
BFS-Primer Camino	23	68	30	88	2.353
BFS-Camino más largo	21	62	29	85	1.971
DFS-Primer Camino	23	68	28	82	3.559
DFS-Camino más largo	22	71	31	91	3.412
Disyuntos	31	91	33	97	1.882

En consecuencia, el método de caminos disyuntos presenta la mejor valoración en cuanto al número de soluciones válidas (siendo muy alto su rendimiento con paquetes simples) y en cuanto al menor número promedio de codificadores presentado en las soluciones obtenidas.

6.4 Resultados de las pruebas según el tipo de soluciones

En la Tabla 6.3 y en la Figura 6.2 se observa que con el método de reducción al grafo multicast unisesión, utilizando caminos disyuntos, se obtiene la mayor cantidad de soluciones conjuntas e individuales. El método de caminos disyuntos presenta 28 mejores soluciones conjuntas, de las cuales 27 corresponden con el envío de paquetes simples y una corresponde con el envío de paquetes combinados (grafo No. 2 de la Tabla 6.3). En cuanto a las soluciones con el envío de paquetes simples desde el nodo fuente, no es el mejor en los grafos 12, 13, 25 y 34 de la Tabla 6.3; sin embargo, son solucionados por el método. En estos grafos, el número de codificadores es mayor que el de la mejor solución en los cuatro casos.

Es importante resaltar en la Tabla 6.3 y en la Figura 6.2, el resultado de las soluciones individuales en caminos disyuntos, ya que es el método que más soluciones presenta. También se destaca que los otros cuatro métodos presentan una solución individual (BFS_PRI y DFS_PRI) o ninguna solución.

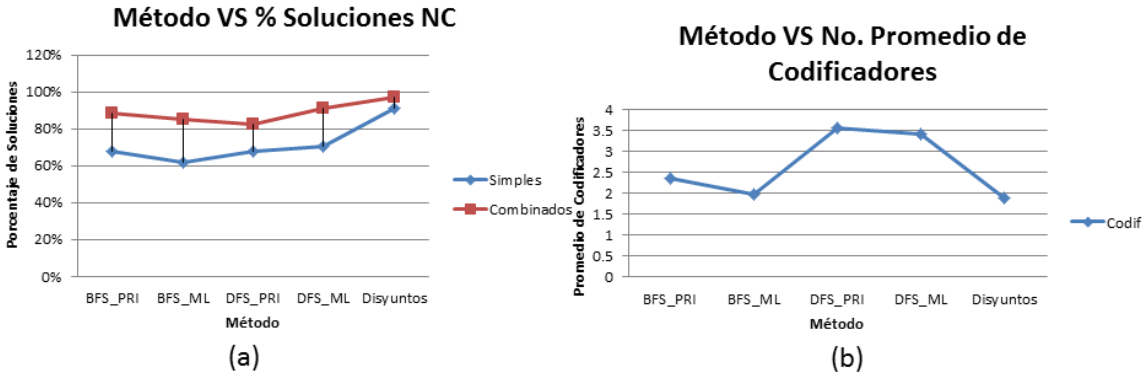


Figura 6.1 Resultados de aplicación de los métodos según (a) Porcentaje de Soluciones con NC, (b) Promedio de Codificadores

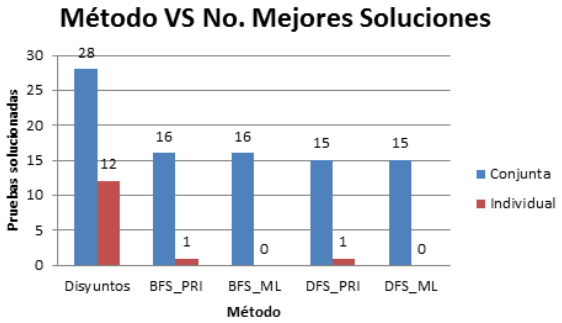


Figura 6.2 Soluciones Conjuntas e Individuales según la aplicación de los métodos

6.5 Rendimiento de los algoritmos según tiempo de cómputo

Los cinco métodos para obtener el grafo multicast de mínimo flujo máximo unisesión G' a partir de un grafo general de comunicaciones G, se probaron con los 37 ejemplos ya antes expuestos en la Tabla 6.3. Según la variable de tiempo de cómputo en la ejecución de los procedimientos, se observa en la Figura 6.3 que todos los métodos tienen un comportamiento exponencial en la medida en que aumentan el número de nodos del grafo de comunicaciones, el número de destinos, el flujo máximo y el número de enlaces.

En la Figura 6.3 se observa que las diferencias en los tiempos de ejecución de los métodos son muy similares para los grafos de comunicaciones con un número de nodos menor o igual a 23. A partir de este punto, las diferencias se van pronunciando de manera acentuada, donde el método de caminos disyuntos se aleja de los demás en gran medida, por el proceso de cómputo que implica obtener los caminos con la mayor calidad, representada en el menor número de codificadores y alcanzar la mayor cantidad de soluciones de los sistemas lineales de ecuaciones que se constituye en cada uno de los nodos sumideros, de tal modo que formen una matriz cuadrada no singular o invertible con los bits de los paquetes entrantes.

6.6 Discusión de los métodos

Según la revisión de los métodos de reducción del grafo de comunicaciones a un grafo de multicast unisesión de mínimo flujo máximo, se puede establecer que el método de búsqueda exhaustiva

basado en caminos disyuntos presenta el mayor número de soluciones factibles a los sistemas lineales de ecuaciones que se configuran en los nodos sumideros luego de la transmisión de los paquetes a través de la red. Este método, además, es el que genera el menor número de nodos de codificación promedio, lo cual redunda en el rendimiento que tendrá la transmisión multicast al momento de implementarla. Por otro lado, el método también es el que mejores soluciones individuales genera por encima de los otros cuatro métodos. La principal desventaja del método es su alto costo computacional por la revisión exhaustiva que lleva a cabo para hallar los caminos de mejor calidad.

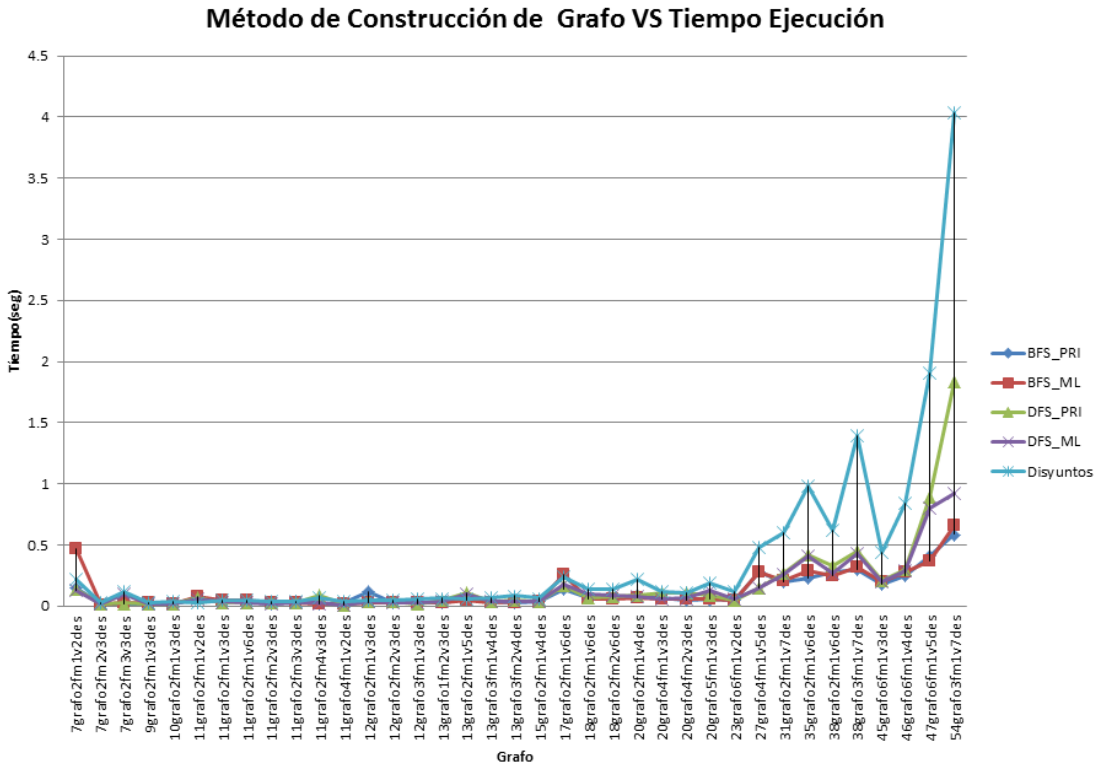


Figura 6.3 Gráfico que representa el Método vs Tiempo de Ejecución

Capítulo 7

Conclusiones y Trabajos Futuros

La investigación llevada a cabo en este trabajo se ha centrado fundamentalmente en dos aspectos a solucionar: en primer lugar, la reducción a un grafo dirigido y acíclico multicast unisesión de mínimo flujo máximo, que permita la resolución de un sistema lineal de ecuaciones bajo la técnica de Network Coding; y en segundo lugar, la obtención del esquema de codificación, que determine la posibilidad de resolver la organización y envío de los paquetes simultáneamente, desde el nodo fuente hacia los nodos sumideros en un grafo multicast de mínimo flujo máximo. De estos dos problemas, el primero presenta muy pocos trabajos desarrollados en la literatura. Esto, en razón de la complejidad que pueden presentar los grafos de comunicaciones para reducirlos a un grafo multicast de enrutamiento unisesión de mínimo flujo máximo, y que permita la recepción simultánea de la cantidad de paquetes que constituyen el flujo.

La búsqueda del grafo multicast unisesión de mínimo flujo máximo, se basa en el teorema establecido por Ahlswede *et al*.[5] a partir de un grafo general dirigido y acíclico de comunicaciones con los nodos fuente y sumideros especificados. En este grafo de comunicación multicast hallado, se podría llevar a cabo la transmisión del mínimo flujo máximo de paquetes en forma simultánea desde el nodo fuente hasta los nodos sumideros.

La resolución del segundo problema, implica el hallazgo, en los nodos sumideros, de los paquetes que solucionan los sistemas de ecuaciones lineales constituidos a lo largo de la transmisión simultánea, utilizando la red multicast unisesión. El número de paquetes enviados y recibidos simultáneamente, constituyen el mínimo flujo máximo común entre el nodo fuente y el conjunto de nodos sumideros.

Ambos problemas se solucionan sobre la base de la técnica matemática de Network Coding y en especial de LNC. Esta técnica permite que se puedan enviar simultáneamente paquetes hasta el mínimo flujo máximo común por los caminos que conducen desde el nodo fuente hacia los nodos sumideros. Varios caminos hacia distintos sumideros pueden converger en nodos de enrutamiento intermedio, configurándolos como nodos codificadores que llevan a cabo combinaciones lineales de sus paquetes entrantes, y reenviando el paquete resultante a través de sus enlaces de salida hacia los nodos destinatarios. A lo largo de los caminos, los paquetes se pueden volver a combinar con otros paquetes en más nodos codificadores hasta alcanzar los nodos sumideros.

Los primeros métodos establecidos en este trabajo para la solución del primer problema planteado, se basan en el método de Ford-Fullkerson aplicando los algoritmos de búsquedas clásicas en grafos: Búsqueda en Anchura (Breadth Firsth Search – BFS) y Búsqueda en Profundidad (Depth First Search – DFS). Ambos métodos se modelaron para establecer el grafo unicast de flujo

máximo entre el nodo fuente y cada nodo sumidero especificado. Luego se igualan los flujos máximos al mínimo valor a través de la eliminación de los primeros caminos o de los caminos más largos en los grafos que sobrepasan el mínimo flujo máximo común. Por último, se logra obtener el grafo multicast de mínimo flujo máximo mediante la mezcla de los grafos unicast de flujos máximos igualados.

En resumen, aplicando Ford-Fullkerson con los dos algoritmos de búsqueda y las dos modalidades de reducción, se obtienen grafos multicast de mínimo flujo máximo, los cuales superan en más de un 60% las resoluciones de ecuaciones lineales en los nodos sumideros para obtener el mínimo flujo máximo de paquetes simples distintos enviados desde el nodo fuente. De igual forma, con paquetes combinados salientes del nodo fuente, las resoluciones de ecuaciones lineales en los nodos sumideros, superan el 80%. Sin embargo, el número de nodos codificadores promedio generados por los dos métodos con sus dos variaciones está por encima de dos, excepto para el método de Búsqueda en Anchura con eliminación del camino más largo (BFS_ML).

El algoritmo, que mejores resultados arroja, está basado en la búsqueda exhaustiva de todos los caminos que conducen desde el nodo fuente hacia cada sumidero. Este algoritmo elimina las rutas más largas y las que más colisiones presentan entre sí. Así obtiene el grafo unicast de flujo máximo entre el nodo fuente y cada sumidero. Posteriormente, se igualan los flujos máximos de los grafos unicast que sobrepasan el flujo máximo común. Se toman los caminos que sobrelapan en la mayor cantidad de nodos y enlaces a los caminos de un grafo unicast que cumpla con el flujo máximo común y que, además, estos caminos sean de la menor longitud posible. Luego, se mezclan los grafos de mínimo flujo máximo común para obtener el grafo multicast unisesión, el cual puede tener todavía caminos que no lleven a una solución del sistema lineal de ecuaciones basado en Network Coding. Ante esto, se eliminan los nuevos caminos no disyuntos creados durante la mezcla, aplicando como principio que la diferencia simétrica de los conjuntos de segundos nodos (asociados a los paquetes salientes del nodo fuente) que arriben a cada nodo sumidero no debe ser el conjunto vacío.

Este algoritmo genera el mayor número de grafos multicast unisesión en los que se soluciona el sistema lineal de ecuaciones en los nodos sumideros, con envío de paquetes simples y combinados desde el nodo fuente. Igualmente, es el algoritmo que presenta el menor promedio de nodos codificadores por grafo. Sin embargo, es el que tiene el mayor tiempo de ejecución porque halla las rutas de mejor calidad, demostrado en que es el que mayor cantidad de mejores soluciones únicas arroja.

El esquema de codificación propuesto, para el ordenamiento de los paquetes de salida por los enlaces salientes desde el nodo fuente y la resolución del sistema lineal de ecuaciones en los nodos sumideros, se basa en la formulación, en los nodos sumideros, de ecuaciones linealmente independientes a partir de la combinación lineal de códigos que etiquetan a los paquetes generados y emergentes por los enlaces salientes desde el nodo fuente. Estas ecuaciones se forman en cada enlace entrante en los nodos sumideros. El número de ecuaciones a resolver en cada nodo sumidero es el mínimo flujo máximo, y la resolución de estas ecuaciones en todos los sumideros, conlleva que el esquema de codificación sea válido, de lo contrario no tiene solución y, por ende, el grafo multicast no puede enviar simultáneamente el mínimo flujo máximo a los sumideros.

Como trabajos futuros a desarrollar están:

Afinamiento al modelo de caminos disyuntos: Afinar el modelo de caminos disyuntos para incrementar la solución de los grafos multicast unisesión, y aumentar aún más su porcentaje de rendimiento en este aspecto. Igualmente, manteniendo la cota más baja en el promedio de nodos codificadores.

Generación de todas las soluciones al sistema lineal de ecuaciones: Modelar un mecanismo que permita encontrar todos los ordenamientos resultantes de la solución al sistema lineal de ecuaciones sobre la base de Network Coding.

Soluciones parciales: Considerar un modelo que presente las soluciones parciales; es decir, que logre entregar los paquetes en los nodos sumideros donde se obtenga la solución. Igualmente, considerar cómo reajustar el grafo multicast, ya sea eliminando nodos o enlaces y/o adicionando nodos o enlaces, del grafo general de comunicaciones, para aumentar el número de soluciones posibles y sin perder la dimensión del mínimo flujo máximo en el grafo multicast original.

Sistema multicast multisesión: Considerar la solución de un sistema que contenga la mezcla de varios grafos multicast unisesión de distintos mínimos flujos máximos, donde existan nodos de codificación y sumideros comunes. Este sistema debe considerar retardos y encolamiento en estos nodos comunes.

Sistema multicast unisesión con pérdidas en los enlaces: Dado que las redes pueden llegar a tener un mal funcionamiento en algunos enlaces, ya sea por ruptura, congestión o cualquier otra circunstancia, es necesario considerar la pérdida de paquetes y las acciones que se deben ejecutar para la recuperación de la información perdida. Es significativo, en esta propuesta, que se lleve a cabo una simulación de la red multicast para comprobar las acciones que logren la solución.

Estudio estadístico de los modelos: Revisar, a través de un diseño de experimentos, el comportamiento de los métodos de generación de grafos multicast de mínimo flujo máximo, dentro del marco de las variables que determinan un grafo general de comunicaciones: número de nodos, número de enlaces, número de sumideros, número de enlaces de salida, y flujo.

Bibliografía

[1] W. R. Parkhurst, *Cisco Multicasting Routing and Switching*. McGraw-Hill, Inc., 1999.

[2] «Usuarios de internet y redes sociales en el mundo en 2018». [En línea]. Disponible en: https://ilifebelt.com/usuarios-internet-redes-sociales-mundo-2018/2018/02/. [Accedido: 18-may-2018].

[3] K. I. Park, *QoS in packet networks*, vol. 779. Springer Science & Business Media, 2004.

[4] W. C. Hardy, *QoS: measurement and evaluation of telecommunications quality of service*. Wiley Chichester, England, 2001.

[5] R. Ahlswede, N. Cai, S.-Y. Li, y R. W. Yeung, «Network information flow», *IEEE Trans. Inf. Theory*, vol. 46, n.º 4, pp. 1204-1216, 2000.

[6] L. R. Ford y D. R. Fulkerson, «Maximal flow through a network», *Class. Pap. Comb.*, pp. 243-248, 1987.

[7] S. I. Grossman, M. G. Osuna, y F. P. Soto, *Álgebra lineal*. Grupo Editorial Iberoamericana, 1983.

[8] V. V. Prasolov, *Problems and theorems in linear algebra*, vol. 134. American Mathematical Soc., 1994.

[9] S. I. Grossman, *Elementary linear algebra*. Brooks/Cole Publishing Company, 1994.

[10] C. C. Pinter, «Set theory», 1976.

[11] J. D. Monk, «Introduction to set theory», 1973.

[12] K. Hrbacek y T. Jech, *Introduction to Set Theory, Revised and Expanded*. Crc Press, 1999.

[13] P. A. Chou y Y. Wu, «Network coding for the internet and wireless networks», *IEEE Signal Process. Mag.*, vol. 24, n.º 5, pp. 77-85, 2007.

[14] T. H. Cormen, *Introduction to algorithms*. MIT press, 2009.

[15] D. E. Knuth, *The art of computer programming. Vol. 1: Fundamental algorithms. Second printing*. Addison-Wesley Publishing Co., Reading, Mass.-London-Don Mills, Ont, 1969.

[16] S. Dasgupta, C. H. Papadimitriou, y U. Vazirani, *Algorithms*. McGraw-Hill, Inc., 2006.

[17] R. Koetter y M. Médard, «An algebraic approach to network coding», *IEEEACM Trans. Netw. TON*, vol. 11, n.º 5, pp. 782-795, 2003.

[18] T. Ho y D. Lun, *Network coding: an introduction*. Cambridge University Press, 2008.

[19] A. Sprintson, «Network coding and its applications in communication networks», en *Algorithms for Next Generation Networks*, Springer, 2010, pp. 343-372.

[20] L. H. Sahasrabuddhe y B. Mukherjee, «Multicast routing algorithms and protocols: A tutorial», *IEEE Netw.*, vol. 14, n.º 1, pp. 90-102, 2000.

[21] S. E. Deering, *Multicast routing in internetworks and extended LANs*, vol. 18. ACM, 1988.

[22] R. Dougherty, C. Freiling, y K. Zeger, «Linearity and solvability in multicast networks», *IEEE Trans. Inf. Theory*, vol. 50, n.º 10, pp. 2243-2256, 2004.

[23] Y. Wu, P. A. Chou, y K. Jain, «A comparison of network coding and tree packing», en *Information Theory, 2004. ISIT 2004. Proceedings. International Symposium on*, 2004, p. 143.

[24] D. Cieslik, *Steiner minimal trees*, vol. 23. Springer Science & Business Media, 2013.

[25] E. N. Gilbert y H. O. Pollak, «Steiner minimal trees», *SIAM J. Appl. Math.*, vol. 16, n.º 1, pp. 1-29, 1968.

[26] F. K. Hwang, D. S. Richards, y P. Winter, *The Steiner tree problem*, vol. 53. Elsevier, 1992.

[27] K. Jain, M. Mahdian, y M. R. Salavatipour, «Packing steiner trees», en *Proceedings of the fourteenth annual ACM-SIAM symposium on Discrete algorithms*, 2003, pp. 266-274.
[28] L. C. Lau, «An approximate max-Steiner-tree-packing min-Steiner-cut theorem», en *Foundations of Computer Science, 2004. Proceedings. 45th Annual IEEE Symposium on*, 2004, pp. 61-70.
[29] B. Y. Wu y K.-M. Chao, «Steiner Minimal Trees», *URL Httpwww Csie Ntu Edu Tw Kmchaotree07sprSteiner PdfLast Accessed 1303 2017*, 2004.
[30] J. Zhang, P. Fan, y K. B. Letaief, «Network coding for efficient multicast routing in wireless ad-hoc networks», *IEEE Trans. Commun.*, vol. 56, n.º 4, 2008.
[31] S.-Y. R. Li, R. W. Yeung, y N. Cai, «Linear network coding», *Inf. Theory IEEE Trans. On*, vol. 49, n.º 2, pp. 371-381, 2003.
[32] B. Li y Y. Wu, «Network coding [scanning the issue]», *Proc. IEEE*, vol. 99, n.º 3, pp. 363-365, 2011.
[33] R. W. Yeung y N. Cai, *Network coding theory*. Now Publishers Inc, 2006.
[34] C. Fragouli, J.-Y. Le Boudec, y J. Widmer, «Network coding: an instant primer», *ACM SIGCOMM Comput. Commun. Rev.*, vol. 36, n.º 1, pp. 63-68, 2006.
[35] S. Katti, H. Rahul, W. Hu, D. Katabi, M. Médard, y J. Crowcroft, «XORs in the air: practical wireless network coding», en *ACM SIGCOMM computer communication review*, 2006, vol. 36, pp. 243-254.
[36] C. Fragouli y E. Soljanin, «Network coding applications», *Found. Trends® Netw.*, vol. 2, n.º 2, pp. 135-269, 2008.
[37] J. Xie, C. Dong, H. Wang, y W. Wang, «A packet-based load balancing scheme for network coding in tactical heterogeneous wireless networks», en *Wireless Communications & Signal Processing (WCSP), 2016 8th International Conference on*, 2016, pp. 1-5.
[38] T. Noguchi, T. Matsuda, y M. Yamamoto, «Performance evaluation of new multicast architecture with network coding», *IEICE Trans. Commun.*, vol. 86, n.º 6, pp. 1788-1795, 2003.
[39] S. Tao, W. Qiao, Z. Yang, y W. Cheng, «Routing algorithm of network coding on multicast», en *Computational Intelligence and Security Workshops, 2007. CISW 2007. International Conference on*, 2007, pp. 354-357.
[40] S.-Y. R. Li, Q. T. Sun, y Z. Shao, «Linear network coding: Theory and algorithms», *Proc. IEEE*, vol. 99, n.º 3, pp. 372-387, 2011.
[41] N. J. Harvey, D. R. Karger, y K. Murota, «Deterministic network coding by matrix completion», en *Proceedings of the sixteenth annual ACM-SIAM symposium on Discrete algorithms*, 2005, pp. 489-498.
[42] N. Jayakumar, S. P. Khatri, K. Gulati, y A. Sprintson, «Network coding for routability improvement in VLSI», en *Proceedings of the 2006 IEEE/ACM international conference on Computer-aided design*, 2006, pp. 820-823.
[43] A. G. Dimakis, P. B. Godfrey, Y. Wu, M. J. Wainwright, y K. Ramchandran, «Network coding for distributed storage systems», *IEEE Trans. Inf. Theory*, vol. 56, n.º 9, pp. 4539-4551, 2010.
[44] A. G. Dimakis, K. Ramchandran, Y. Wu, y C. Suh, «A survey on network codes for distributed storage», *Proc. IEEE*, vol. 99, n.º 3, pp. 476-489, 2011.
[45] Z. Yu, Y. Wei, B. Ramkumar, y Y. Guan, «An efficient signature-based scheme for securing network coding against pollution attacks», en *INFOCOM 2008. The 27th Conference on Computer Communications. IEEE*, 2008.
[46] H. Balli, X. Yan, y Z. Zhang, «Error correction capability of random network error correction codes», en *2007 IEEE International Symposium on Information Theory*, 2007, pp. 1581-1585.
[47] D. Charles, K. Jain, y K. Lauter, «Signatures for network coding», *Int. J. Inf. Coding Theory*, vol. 1, n.º 1, pp. 3-14, 2009.
[48] R. Dougherty, C. Freiling, y K. Zeger, «Networks, matroids, and non-Shannon information inequalities», *IEEE Trans. Inf. Theory*, vol. 53, n.º 6, pp. 1949-1969, 2007.

[49] S. El Rouayheb, A. Sprintson, y C. Georghiades, «On the index coding problem and its relation to network coding and matroid theory», *IEEE Trans. Inf. Theory*, vol. 56, n.º 7, pp. 3187-3195, 2010.
[50] R. W. Yeung, «Network coding theory: An introduction», *Front. Electr. Electron. Eng. China*, vol. 5, n.º 3, pp. 363-390, 2010.
[51] S.-Y. R. Li y Q. T. Sun, «Network coding theory via commutative algebra», *IEEE Trans. Inf. Theory*, vol. 57, n.º 1, pp. 403-415, 2011.
[52] D. S. Lun *et al.*, «Minimum-cost multicast over coded packet networks», *IEEE Trans. Inf. Theory*, vol. 52, n.º 6, pp. 2608-2623, 2006.
[53] J. Schalkamp, «Minimum-Cost Multicast over Coded Packet Networks», *Innov. Internet Technol. Mob. Commun. IITM*, vol. 46, 2012.
[54] C. Zhao, X. Lin, y C. Wu, «The streaming capacity of sparsely connected P2P systems with distributed control», *IEEEACM Trans. Netw. TON*, vol. 24, n.º 1, pp. 58-71, 2016.
[55] L. Keller, A. Le, B. Cici, H. Seferoglu, C. Fragouli, y A. Markopoulou, «MicroCast: cooperative video streaming on smartphones», en *Proceedings of the 10th international conference on Mobile systems, applications, and services*, 2012, pp. 57-70.
[56] M. Yang y Y. Yang, «Peer-to-peer file sharing based on network coding», en *Distributed Computing Systems, 2008. ICDCS'08. The 28th International Conference on*, 2008, pp. 168-175.
[57] T. Small, B. Li, y B. Liang, «Topology affects the efficiency of network coding in peer-to-peer networks», en *Communications, 2008. ICC'08. IEEE International Conference on*, 2008, pp. 5591-5597.
[58] K. Wei, X. Liang, y K. Xu, «A survey of social-aware routing protocols in delay tolerant networks: applications, taxonomy and design-related issues», *IEEE Commun. Surv. Tutor.*, vol. 16, n.º 1, pp. 556-578, 2014.
[59] S. Ahmed y S. S. Kanhere, «HUBCODE: hub-based forwarding using network coding in delay tolerant networks», *Wirel. Commun. Mob. Comput.*, vol. 13, n.º 9, pp. 828-846, 2013.
[60] W. Guan y K. R. Liu, «Mitigating error propagation for wireless network coding», *IEEE Trans. Wirel. Commun.*, vol. 11, n.º 10, pp. 3632-3643, 2012.
[61] S. C. Liew, S. Zhang, y L. Lu, «Physical-layer network coding: Tutorial, survey, and beyond», *Phys. Commun.*, vol. 6, pp. 4-42, 2013.
[62] K. Miller, T. Biermann, H. Woesner, y H. Karl, «Network coding in passive optical networks», en *Network Coding (NetCod), 2010 IEEE International Symposium on*, 2010, pp. 1-6.
[63] K. Fouli, M. Maier, y M. Medard, «Network coding in next-generation passive optical networks», *IEEE Commun. Mag.*, vol. 49, n.º 9, 2011.
[64] M. Belzner y H. Haunstein, «Network coding in passive optical networks», en *Optical Communication, 2009. ECOC'09. 35th European Conference on*, 2009, pp. 1-2.
[65] S. A. Aly y A. E. Kamal, «Network coding-based protection against link failures», en *Proc. IEEE Global Commun.*, 2008, pp. 1-6.
[66] J. K. Sundararajan, M. Médard, M. Kim, A. Eryilmaz, D. Shah, y R. Koetter, «Network coding in a multicast switch», en *INFOCOM 2007. 26th IEEE International Conference on Computer Communications. IEEE*, 2007, pp. 1145-1153.
[67] S. Yang, X. Wang, y B. Li, «Haste: practical online network coding in a multicast switch», en *INFOCOM, 2010 Proceedings IEEE*, 2010, pp. 1-5.
[68] M. A. Marsan, A. Bianco, P. Giaccone, E. Leonardi, y F. Neri, «Multicast traffic in input-queued switches: optimal scheduling and maximum throughput», *IEEEACM Trans. Netw. TON*, vol. 11, n.º 3, pp. 465-477, 2003.
[69] R. W. Yeung, «Network coding: A historical perspective», *Proc. IEEE*, vol. 99, n.º 3, pp. 366-371, 2011.
[70] P. Elias, A. Feinstein, y C. Shannon, «A note on the maximum flow through a network», *IRE Trans. Inf. Theory*, vol. 2, n.º 4, pp. 117-119, 1956.

[71] G. Dantzig y D. R. Fulkerson, «On the max flow min cut theorem of networks», *Linear Inequalities Relat. Syst.*, vol. 38, pp. 225-231, 2003.
[72] R. K. Ahuja, T. L. Magnanti, y J. B. Orlin, «Network flows: theory, algorithms, and applications», 1993.
[73] N. Zadeh, «Theoretical efficiency of the Edmonds-Karp algorithm for computing maximal flows», *J. ACM JACM*, vol. 19, n.º 1, pp. 184-192, 1972.
[74] J. Edmonds y R. M. Karp, «Theoretical improvements in algorithmic efficiency for network flow problems», *J. ACM JACM*, vol. 19, n.º 2, pp. 248-264, 1972.
[75] S. Even, *The max flow algorithm of Dinic and Karzanov: an exposition*. Massachusetts Institute of Technology. Laboratory for Computer Science, 1976.
[76] Z. Galil, «On the theoretical efficiency of various network flow algorithms», *Theor. Comput. Sci.*, vol. 14, n.º 1, pp. 103-111, 1981.
[77] R. K. Ahuja y J. B. Orlin, «A fast and simple algorithm for the maximum flow problem», *Oper. Res.*, vol. 37, n.º 5, pp. 748-759, 1989.
[78] A. KARZANOV, «Determining the maximal flow in a network by the method of preflows», en *Soviet Math. Dokl.*, 1974, vol. 15, pp. 434-437.
[79] A. V. Goldberg, *A new max-flow algorithm*. Laboratory for Computer Science, Massachusetts Institute of Technology, 1985.
[80] A. V. Goldberg y R. E. Tarjan, «A new approach to the maximum-flow problem», *J. ACM JACM*, vol. 35, n.º 4, pp. 921-940, 1988.
[81] R. Lidl y H. Niederreiter, *Finite fields*, vol. 20. Cambridge university press, 1997.
[82] R. Koetter y M. Médard, «Beyond routing: An algebraic approach to network coding», en *INFOCOM 2002. Twenty-First Annual Joint Conference of the IEEE Computer and Communications Societies. Proceedings. IEEE*, 2002, vol. 1, pp. 122-130.
[83] T. Ho, R. Koetter, M. Medard, D. R. Karger, y M. Effros, «The benefits of coding over routing in a randomized setting», 2003.
[84] S. Jaggi, P. A. Chou, y K. Jain, «Low complexity optimal algebraic multicast codes», en *Proc. IEEE Int'l Symp. Information Theory, Yokohama, Japan*, 2003.
[85] S. Jaggi *et al.*, «Polynomial time algorithms for multicast network code construction», *Inf. Theory IEEE Trans. On*, vol. 51, n.º 6, pp. 1973-1982, 2005.
[86] P. Sanders, S. Egner, y L. Tolhuizen, «Polynomial time algorithms for network information flow», en *Proceedings of the fifteenth annual ACM symposium on Parallel algorithms and architectures*, 2003, pp. 286-294.
[87] T. Ho, D. R. Karger, M. Médard, y R. Koetter, «Network coding from a network flow perspective», en *Information Theory, 2003. Proceedings. IEEE International Symposium on*, 2003, p. 441.
[88] A. Tavory, M. Feder, y D. Ron, «Bounds on linear codes for network multicast», en *Electronic Colloquium on Computational Complexity (ECCC)*, 2003, vol. 10.
[89] A. R. Lehman y E. Lehman, «Complexity classification of network information flow problems», en *Proceedings of the fifteenth annual ACM-SIAM symposium on Discrete algorithms*, 2004, pp. 142-150.
[90] Y. Wu, P. A. Chou, y S.-Y. Kung, «Minimum-energy multicast in mobile ad hoc networks using network coding», *IEEE Trans. Commun.*, vol. 53, n.º 11, pp. 1906-1918, 2005.
[91] E. Erez y M. Feder, «Convolutional network codes», en *Information Theory, 2004. ISIT 2004. Proceedings. International Symposium on*, 2004, p. 146.
[92] L. Wu y K. Curran, «A practical network coding and routing scheme based on maximum flow combination», *Int. J. Netw. Manag.*, vol. 22, n.º 5, pp. 373-396, 2012.
[93] T. Shaoguo, H. Jiaqing, Y. Zongkai, R. S. Youail, y C. Wenqing, «Routing algorithm for network coding based multicast», en *Convergence Information Technology, 2007. International Conference on*, 2007, pp. 2091-2095.

[94] T. Ho *et al.*, «A random linear network coding approach to multicast», *IEEE Trans. Inf. Theory*, vol. 52, n.º 10, pp. 4413-4430, 2006.
[95] L. Zosin y S. Khuller, «On directed Steiner trees», en *Proceedings of the thirteenth annual ACM-SIAM symposium on Discrete algorithms*, 2002, pp. 59-63.
[96] R. Diestel, «Graph theory. 2005», *Grad Texts Math*, vol. 101, 2005.
[97] Y. Zhu, B. Li, y J. Guo, «Multicast with network coding in application-layer overlay networks», *IEEE J. Sel. Areas Commun.*, vol. 22, n.º 1, pp. 107-120, 2004.
[98] Y. Chu, S. G. Rao, S. Seshan, y H. Zhang, «A case for end system multicast», *IEEE J. Sel. Areas Commun.*, vol. 20, n.º 8, pp. 1456-1471, 2002.
[99] J. Schmerl, «Minimal spanning trees», *Proc. Am. Math. Soc.*, vol. 132, n.º 2, pp. 333-340, 2004.
[100] C. Nash-Williams, «Edge-Disjoint Spanning Trees of Finite Graphs», *J. Lond. Math. Soc.*, vol. 1, n.º 1, pp. 445-450, 1961.
[101] P. Jayawant y K. Glavin, «Minimum spanning trees», *Involve J. Math.*, vol. 2, n.º 4, pp. 439-450, 2009.
[102] B. Y. Wu y K.-M. Chao, *Spanning trees and optimization problems*. CRC Press, 2004.
[103] D. Tang, X. Lu, y J. Li, «Multicast routing algorithm based on network coding», en *International Conference on Brain Inspired Cognitive Systems*, 2013, pp. 348-357.
[104] L. Wu y K. Curran, «Practical Network Coding Scheme Based on Maximum Flow Combination and Coding Node Identification», en *Internet Technology and Applications, 2010 International Conference on*, 2010, pp. 1-4.
[105] H. Takahashi, «An approximate solution for the Steiner problem in graphs», *Math Jpn.*, vol. 6, pp. 573-577, 1990.
[106] C. K. Constantinou y G. Ellinas, «A novel multicast routing algorithm and its application for protection against single-link and single-link/node failure scenarios in optical WDM mesh networks», *Opt. Express*, vol. 19, n.º 26, pp. B471-B477, 2011.
[107] P. A. Chou, Y. Wu, y K. Jain, «Practical network coding», en *Proceedings of the annual Allerton conference on communication control and computing*, 2003, vol. 41, pp. 40-49.
[108] M. Langberg, A. Sprintson, y J. Bruck, «The encoding complexity of network coding», *IEEE Trans. Inf. Theory*, vol. 52, n.º 6, pp. 2386-2397, 2006.
[109] G. Kirchhoff, «Ueber die Auflösung der Gleichungen, auf welche man bei der Untersuchung der linearen Vertheilung galvanischer Ströme geführt wird», *Ann. Phys.*, vol. 148, n.º 12, pp. 497-508, 1847.
[110] J. Kleinberg y E. Tardos, *Algorithm design*. Pearson Education India, 2006.
[111] R. J. Wilson, *An introduction to graph theory*. Pearson Education India, 1970.
[112] K. Menger, «Zur allgemeinen Kurventheorie», *Fundam. Math.*, vol. 10, n.º 1, pp. 96-115, 1927.
[113] S. Beamer, K. Asanović, y D. Patterson, «Direction-optimizing breadth-first search», *Sci. Program.*, vol. 21, n.º 3-4, pp. 137-148, 2013.
[114] R. Zhou y E. A. Hansen, «Breadth-first heuristic search», *Artif. Intell.*, vol. 170, n.º 4-5, pp. 385-408, 2006.
[115] R. Tarjan, «Depth-first search and linear graph algorithms», en *Switching and Automata Theory, 1971., 12th Annual Symposium on*, 1971, pp. 114-121.
[116] «GitHub - Z3Prover/z3: The Z3 Theorem Prover». [En línea]. Disponible en: https://github.com/Z3Prover/z3. [Accedido: 08-jun-2017].
[117] «Welcome to Python.org», *Python.org*. [En línea]. Disponible en: https://www.python.org/. [Accedido: 31-may-2018].
[118] S. Riis, «Linear versus non-linear boolean functions in network flow», en *38th Annual Conference on Information Science and Systems (CISS), Princeton, NJ*, 2004.
[119] «MathWorks - Makers of MATLAB and Simulink». [En línea]. Disponible en: https://www.mathworks.com/. [Accedido: 24-may-2017].